PARAMETER ESTIMATION USING EXTENDED KALMAN FILTER AND ULTRASONIC PULSE TIME OF FLIGHT TO LOCATE TRANSIENT, CONCENTRATED HEATING SOURCES

By

Michael R. Myers

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Mechanical Engineering

May, 2012

Nashville, Tennessee

Approved:

Professor Greg Walker, Chair

Professor Ariosto Jorge

Professor Robert Pitz

Professor Al Strauss

Professor Mitch Wilkes

To wife, Cindy, and our boys, Mackenzie and Stuart;

to my mom;

and to my dad, I wish he was here to read this and celebrate.

ACKNOWLEDGMENTS

I would like to acknowledge my advisor, Professor Greg Walker for his assistance, guidance, and teaching throughout my Ph.D. career; Professor Ariosto Jorge for his mentoring, collaboration, and friendship; Professor Mitch Wilkes for his assistance in understanding recursive state estimation and his view of the world; Professor Al Strauss for his early morning discussions and breadth and depth of knowledge; Professor Bob Pitz for his encouragement to pursue a Ph.D. after such a significant time away from academia; and Dr. Don Yuhas and Mr. Mark Mutton from Industrial Measurement Systems, Inc. for their experimental support and patience as we progressed through discovery and problem solving. I would also like to acknowledge the financial support from the the United Stated Air Force Research Laboratory (AFRL) and Vanderbilt University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

A	area (m^2);
	amplitude (m);
	state Jacobian
a	state model;
	material thickness
B	measurement Jacobian
b	expected measurement
C	heat capacity (J/K)
c_p	specific heat (J/kg K)
c	distance from ellipse center to ellipse edge along the major axis
d	distance from ellipse center to ellipse edge along the minor axis
E	energy (J);
	Young's modulus (GPa)
F	cumulative density function
G	ultrasonic time of flight (s)
$\overline{G}$	expected ultrasonic time of flight (s)
$gain$	particle filter gain
h	convection heat transfer coefficient (W/m^2 K)
I_n	$n \times n$ identity matrix
K	Kalman gain
k	thermal conductivity (W/m K)
L	length (m)
l	length (m)
M	adaptive extended Kalman filter gain
m	number of particles
Nu	Nusselt number
P	probability
Pr	Prandtl number
p	probability
Q	heat source;
	state model covariance
q''	heat flux (W/m^2)
R	thermal resistance (K/W);
	measurement covariance;
	radius (m)
Ra	Rayleigh number
r	radius (m)
$\bar{r}$	expected radius (m)

S	sensitivity
T	temperature (K)
t	time (s)
U	control input
v	sound speed (m/s)
w	width (m);
	particle weight
X	state
$\overline{X}$	predicted state
x, y, z	rectangular coordinates (m)
$\bar{x}, \bar{y}$	predicted rectangular coordinates (m)
Z	actual measurements

Greek letters

α	thermal diffusivity (m^2/s)
β	volumetric thermal expansion (1/K);
γ	temperature dependence of Young's modulus (1/K)
Δ	normalized temperature difference
∇	Laplacian operator
δ	coefficient
θ	temperature change relative to reference (K)
$\overline{\theta}$	predicted temperature change relative to reference (K)
κ	thermal conductivity (W/m K)
ξ	ultrasonic time of flight temperature factor (1/K)
ρ	density (kg/m^3)
Σ	state covariance
$\overline{\Sigma}$	predicted covariance
σ^2	variance for a Gaussian probability density function
τ	period (s)
ϕ	information vector
$\overline{\phi}$	predicted information vector
Ω	information matrix
$\overline{\Omega}$	predicted information matrix

Subscripts

0	initial or zero point
amb	ambient
i	sensor
inf	infinity

j sensor
g Gaussian profile
q heat source
s heat source
T temperature
t time (s)
ts time-scaling coefficient

β state parameters vector

CHAPTER I

INTRODUCTION

The objective of this dissertation is to estimate the location and characteristics of concentrated heating sources on a surface. The heating sources are transient as they may be moving and varying temporally in magnitude. The medium being heated is a solid material although the methods developed in this work could be applied to heating of gases and liquids. This work is applicable to numerous industries including manufacturing and aerospace. The applications include a sensor and estimation system for manufacturing processes, test vehicles, operational vehicles, and durable goods.

Primary motivation for, and the funding that initiated, this work came from the aerospace industry. Knowledge of where air flowing across a body transitions from laminar flow to turbulent flow (Figure 1.1) can provide numerous benefits to air vehicle design, thermal protection system design, and air vehicle in-flight control [Reed et al., 1997]. Of particular interest in this work is the transition region for hypersonic vehicles (Mach 5+). At the transition between laminar and turbulent flow, a change in body-surface temperature has been measured for hypersonic conditions [Horvath et al., 2002, Schneider, 1999, Schneider, 2004, Berger et al., 2009] and is illustrated in Figure 1.2. Thus, a measurement system is envisioned that leverages the hypersonic body-surface heating profile to locate the boundary layer transition region.

The need for such a system arises as active scramjet and hypersonic research programs are conducted by numerous countries including the United States, Australia, Brazil, Russia, India, France, and Germany. Government agencies funding scramjet and hypersonic research include the US National Aeronautics and Space Administration (NASA), the US Air Force Research Laboratory (AFRL), the US Defense Advanced Research Projects Agency (DARPA), the US Office of Naval Research (ONR), the Australian Defence Science and Technology Organisation (DSTO), the Brazilian Air Force (FAB), the Russian Central Institute for Aviation Motors development (CIAM), the Indian Space Research Organisation (ISRO), the French aerospace research center (ONERA), and the German Research Foundation (DFG). Past and present scramjet and hypersonic research programs include NASA's X-15, X-30 National Aero-Space Plane (NASP), X-43A, and X-51, DSTO and AFRL's Hypersonic International Flight Research Experimentation (HiFiRE), DSTO and DARPA's Hypersonic Collaborative Australia/United States Experiment (HyCAUSE), FAB's 14X, CIAM's GLL Holod and GLL Igla, the Russian Defense Ministry's Tupolev Tu-2000, and ISRO's hypersonic flight experiment (HEX).

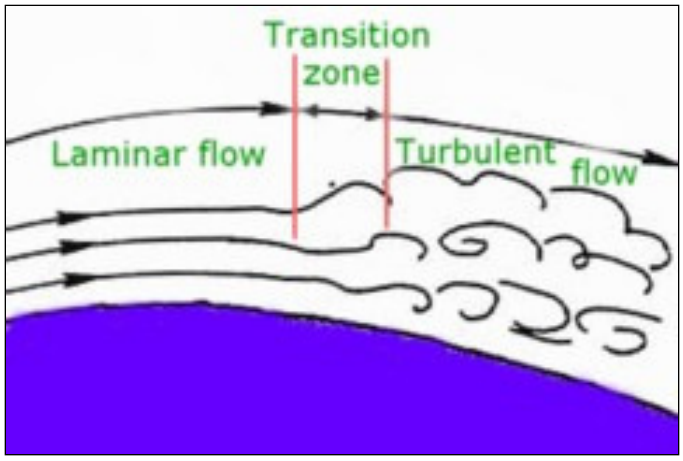

Figure 1.1 Flow regimes [Recreational Aviation Australia, 2010].

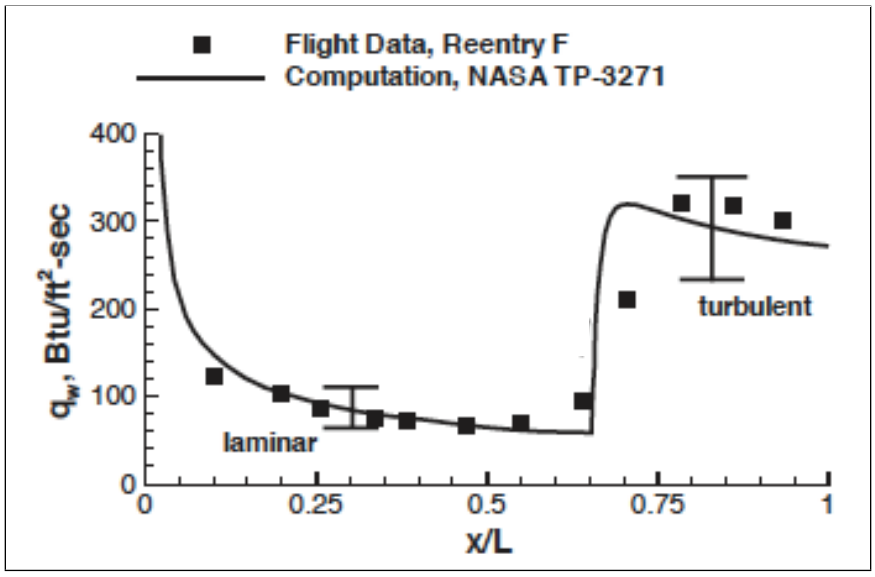

Figure 1.2 Heating profile on a ballistic RV, peak Mach=20 [Schneider, 2004].

1.1 Contributions to science

This dissertation and the underlying research and published works contribute to science in the following areas:

- Employing the extended Kalman filter for a transient heat transfer problem [Myers et al., 2010b, Myers et al., 2012a, Myers et al., 2012d]. Kalman filters are traditionally used in navigation [Bar-Shalom and Li, 2001], guidance [Lin, 1991], speech enhancement [Paliwal, K.K. and Basu, Anjan, 1987], weather forecasting [Burgers et al., 1998], and other non-heat transfer related problems.
- Incorporating a 3D model into the inverse method, instead of a 1D or 2D model [Myers et al., 2012g].
- Exploring the sensitivity of ultrasound sensors to heating source location, boundary conditions, and thermal conductivity [Myers et al., 2012c, Myers et al., 2011].
- Addressing uncertainty by tailoring the state model covariance in the extended Kalman filter during each iteration to achieve more robust performance [Myers et al., 2012f].
- Detailing an innovative measurement model that produces a closed-form Jacobian [Myers et al., 2012b].

1.2 Measurement strategies

Directly observing and measuring the transition region in an operational vehicle is difficult because the harsh environment presents numerous challenges including high-speed airflow and high surface temperatures [Fay and Riddell, 1958, Kendall, 1975, Schook et al., 2001, Gai and Hayne, 2010]. The mechanisms leading to transition are still poorly understood [Bertin and Cummings, 2003, Malik, 1989]. Ground tests of the boundary layer transition generally produce unsatisfactory results due to the high noise levels of ground facilities compared to actual flight conditions [Kimmel et al., 2007]. Sound fields radiated from the turbulent tunnel-wall boundary layers in ground facilities are the sources of this undesirable noise.

Several temperature measurement strategies have been studied and employed including using thermocouples [Frankel et al., 2010], thermopiles [Pullins and Diller, 2010], thin-film temperature gauges [Walker et al., 2000], optical sensors coupled with thermographic phosphors [Yu et al., 2010, Liu et al., 2010], infrared sensors [Gauffre, 1988], and ultrasonic transducers. A popular temperature sensor is the thermocouple, which can be used to measure temperature differences or transient changes in temperature. Normally, thermocouples 1) have a thermal mass that limits

measurement of high frequency components, 2) require tedious calibration, and 3) are intrusive devices that disturb the phenomena being measured. Two of ten planned HiFiRE flight experiments have been conducted where the test vehicle was carried some 200 kilometers into the atmosphere aboard a rocket launched from the Woomera test range in Australia. The test vehicle then dives back into the atmosphere at high speed to test the hypersonic flight technology. Researchers in the HiFiRE program are employing Medtherm dual junction coaxial thermocouples to get surface and back wall temperature measurement versus time which is processed through an inverse solver employing a 1D or 2D forward conduction solution to obtain heat flux versus time [Kimmel et al., 2007]. The dual junction coaxial thermocouples are small in size (standard diameter is 0.061 inches) and consist of a central wire surrounded by a coaxial insulating annulus and an outer annulus of a dissimilar metal. The gauge is inserted in a through-hole in the test article and the thermocouple junction is created at the test article surface by filing or sanding the end of the gauge to drag metal from the annulus over the central wire. The thermocouples are attached to the aeroshell with a 5 cm separation and are sampled at 400Hz during flight. Data is recorded and analyzed post-flight.

In this work, ultrasonic sensing is explored as a more robust measurement strategy along with an inversion method employing the extended Kalman filter for transition region localization and characterization in real-time. Unlike other temperature measurement devices, the ultrasonic sensor is located away from the boundary of interest. Consequently, the phenomenon that is being measured is not disturbed. In addition, the sample rate is limited only by the speed of sound through the medium, and the temperature is proportional to an easily measured quantity, namely, time of flight [Marioli et al., 1992]. Ultrasonic pyrometry has proven effective for gases, fluids, and solids as long as direct access to the material where the temperature being measured is available [Lynnworth and Papadakis, 1970, Wadley et al., 1986, Tasman et al., 1977]. Furthermore, ultrasonic pyrometry has been used in many process control systems [Konno et al., 1993, Hoyle and Luke, 1994] and in non-destructive evaluation and defect detection for decades with a great deal of success [Yee and Couchman, 1976, Moll et al., 2010, Zhu et al., 2010]. Therefore, many technological advances in ultrasonic thermometry exist that we can leverage for measurement of heating loads.

In addition to employing ultrasonic sensors instead of thermocouples, the solution envisioned in this work involves an inverse procedure based on the extended Kalman filter. Kalman filters construct a framework of predicting the state based on an input to the system and correcting the predicted state based on sensor observations [Majji et al., 2010, Bertsekas, 1996]. Kalman filters were invented by Peter Swerling (1958) and Rudolf Kálmán (1960) as a technique for filtering and prediction in linear Gaussian systems [Thrun et al., 2006]. Kalman filters have been used extensively in guidance and navigation systems [Thrun et al., 2006], have been used in various

state estimation scenarios [Rochinha and Peirce, 2010], but have not seen much activity in the heat transfer realm [Vianna et al., 2009]. While this work concentrates on the extended Kalman filter, consideration and comparison is made with the particle filter [Thrun et al., 2006, Vianna et al., 2010] and least squares [Woodbury, 2003a].

1.3 Ultrasonic thermometry

In general, boundary temperatures can be recovered from ultrasonic pulse data from an inversion routine operating on a suitable conduction model. The time of flight for a pulse through an isothermal, homogeneous medium is given as $G = L/v$, where L is the distance between transducers and v is the speed of sound through the medium. Figure 1.3 illustrates a pulse-echo time of flight measurement where one transducer sends a pulse, the pulse is reflected off the medium's boundary, and the same transducer receives the pulse. In the pulse-echo arrangement, L is twice the thickness of the medium ($L = 2a$). Figure 1.4 illustrates a one-way time of flight measurement, sometimes referred to as pitch-catch, where two transducers are used. For the one-way scenario, L is the distance between transducers. One transducer sends a pulse, the pulse travels through the medium, and a second transducer spatially separated from the sending transducer receives the pulse. If the medium contains temperature variations, then the time of flight is obtained by integrating the temperature-dependent sound speed over the thickness.

$$G = \int_0^L \frac{1}{v(T(x))}\,dx = \frac{1}{v_o}\int_0^L [1 + \xi\,\theta(x)]\,dx \tag{1.1}$$

In the foregoing expression, v_o is the sound speed at some reference temperature T_o, $\theta(x) = T(x) - T_o$ is the change in temperature relative to the reference, and ξ is the time of flight time temperature factor. In actuality, the length and velocity both change with temperature—the length because of the coefficient of thermal expansion (CTE, denoted as α here) and the velocity because the lattice stiffness is a function of temperature for anharmonic crystals. However, the two mechanisms are correlated for our purposes, which means the effects cannot be distinguished when considering time of flight data. Consequently, in defining time of flight, the two effects are lumped into the parameter ξ.

The parameter ξ has a physical interpretation that can be seen from its definition.

$$\xi \equiv \frac{1}{G}\frac{dG}{dT} = \frac{1}{L}\frac{dL}{dT} - \frac{1}{v}\frac{dv}{dT}. \tag{1.2}$$

The first term on the right hand side is the linear coefficient of thermal expansion for isotropic materials (α). The temperature dependence of the velocity can be approximated by writing the

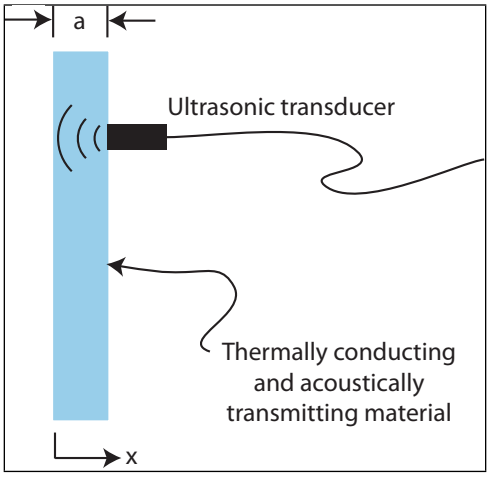

Figure 1.3 Ultrasonic pulse-echo technique.

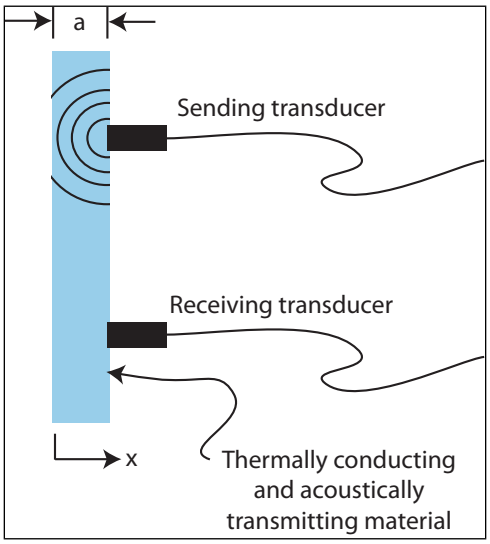

Figure 1.4 One-way ultrasonic pulse technique.

sound speed in a solid in terms of the Young's modulus and density ($v \equiv \sqrt{E/\rho}$), such that

$$\frac{1}{v}\frac{dv}{dT} = \frac{1}{2}\left[\frac{1}{E}\frac{dE}{dT} - \frac{1}{\rho}\frac{d\rho}{dT}\right]. \tag{1.3}$$

The second term on the right hand side in equation 1.3 is the volumetric coefficient of thermal expansion given as -3α for isotropic materials [Turcotte and Schubert, 2002]. The temperature dependence on the Young's modulus is considered a constant [Liu, 1984, Augereau et al., 2007], which we will define as γ. Now our time of flight factor becomes

$$\xi = -\frac{1}{2}(\alpha + \gamma). \tag{1.4}$$

The negative sign in equation 1.4 may seem counter-intuitive because the time of flight should increase with temperature if we consider only thermal expansion. However, we recognize that a decrease in density (second term on the right hand side in equation 1.3) will increase velocity at a rate greater than the decrease resulting from linear thermal expansion. Normally the stiffness of a material will decrease with temperature, therefore $\gamma < 0$ and, in general, ξ is positive. If ξ is constant, the time of flight can be written as

$$G = \frac{L}{v_0} + \frac{2\xi}{v_0}\int_0^L \theta(x)\,dx \tag{1.5}$$

and can be simplified to

$$G = \frac{L}{v_o}\left(1 + \xi\,\theta_{avg}\big|_0^L\right). \tag{1.6}$$

Measuring time of flight can be accomplished by sampling the signal received by the ultrasonic transducers and employing cross-correlation techniques [Hein, 1993, Loupas, 1995]. Figure 1.5 illustrates a typical signal received from an ultrasonic pulse. Cross-correlation depends upon identifying a repeating pattern in the signal and then measuring the time between sending the pulse and receiving the pattern. Three windows have been identified in Figure 1.5 as candidates for the cross-correlation pattern. One of these windows is chosen based on repeatability of the pattern and the ability to recognize the pattern in subsequent measurements.

1.4 Parameter estimation

Figure 1.6 illustrates the conceptual interaction of a hypersonic vehicle with its environment. The environment is a dynamic system that possesses an internal state. For this work, it is convenient to think of the state as the collection of all the vehicle's aspects and its environment that can affect the future. Certain state variables can change over time such as the location of the

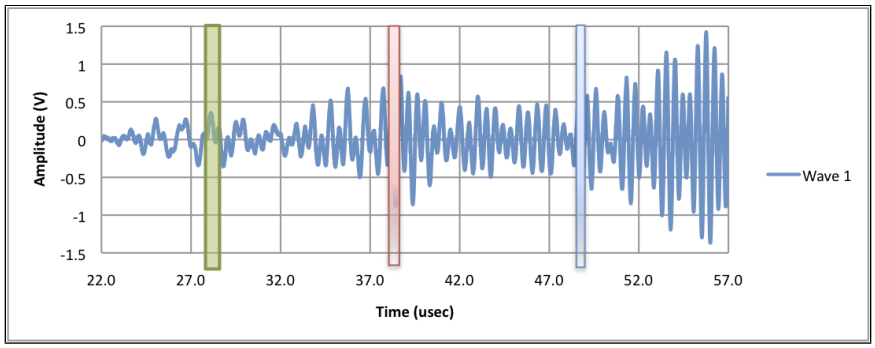

Figure 1.5 Representative sample of the received signal from an ultrasonic pulse illustrating three possible cross-correlation windows. Material is 0.635 cm thick stainless steel 316L with 2 MHz transducers spaced approximately 8 cm apart.

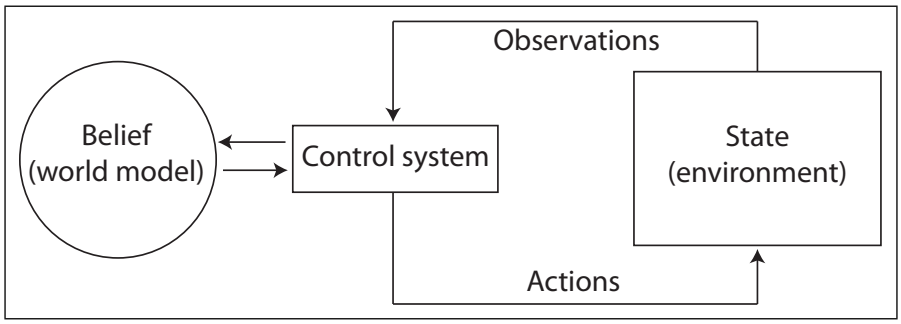

Figure 1.6 Hypersonic vehicle environment interaction.

boundary layer transition, vehicle angle of attack, Mach number, and Reynolds number. Other state variables tend to remain static, such as the location of the ground or the earth. There are endless possibilities for potential state variables. Throughout this work, state will be denoted X and the specific variables included in the state will depend upon the context. The state at time t will be denoted X_t. Time is considered discrete, that is, all interesting events take place at discrete time steps $t = 0, 1, 2, \ldots$.

There are two fundamental types of interactions between a hypersonic vehicle and its environment: The vehicle can influence the state of its environment through its control system, and it can gather information about the state through its sensors. Control actions include moving attitude control surfaces and changing fuel flow rate. Sensors are noisy and many parameters cannot be measured directly. As a consequence, the vehicle maintains an internal belief with regards to its state. The distinction between measurement and control is crucial. While measurements over

time tend to increase the vehicle's knowledge, motion and control inputs tend to induce a loss of knowledge due to the inherent noise in control actuation and the stochasticity of the environment.

Parameter estimation addresses the problem of estimating quantities that are not directly observable but that can be inferred from sensor data. Sensors carry only partial information and their measurements are corrupted by noise. Parameter estimation seeks to recover state variables from the sensor data. Probabilistic parameter estimation algorithms compute belief distributions over possible world states. A state X_t is called complete if it is the best predictor of the future. Completeness entails that knowledge of past states, measurements, and control inputs carry no additional information that would help us predict the future more accurately. In practice, it is impossible to specify a complete state for any realistic heat transfer situation. A complete state includes not only aspects of the object itself and of the environment immediately surrounding the object being studied but also the environment away from the object that may affect its future. Some of these elements are hard to obtain and therefore practical implementations single out a small subset of all state variables.

The Markov assumption postulates that past and future data are independent if one knows the current state X_t. Unmodeled environment dynamics, inaccuracies in probabilistic models, and approximation errors induce violations of the Markov assumption. In principle, many of these variables can be included in the state, however, incomplete state representations are often preferable to more complete ones to reduce the computational complexity of the filter algorithm. It is advisable to exercise care when defining the state X_t so that the effect of unmodeled state variables has close to random effects.

Four recursive state estimation methods are examined in this work: extended Kalman filter, information filter, particle filter, and ordinary least squares. The extended Kalman filter is the preferred method, as will be discussed later, however the other methods are included for performance comparison.

1.4.1 Extended Kalman filter

The extended Kalman filter is a member of a family of recursive state estimators collectively called Gaussian filters. Historically, Gaussian filters constitute the earliest tractable implementations of the Bayes filter for continuous spaces [Thrun et al., 2006]. Kalman filters construct a framework of predicting the state based on an input to the system and correcting the predicted state based on sensor observations. Kalman filters were invented by Swerling (1958) and Kalman (1960) as a technique for filtering and prediction in linear Gaussian systems [Thrun et al., 2006]. Kalman filters assume that all continuous random variables possess probability density functions (PDFs). A common density function is that of the one-dimensional normal distribution with mean

μ and variance σ^2. The PDF of a normal distribution is given by the following Gaussian function:

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{1.7}$$

Normal distributions play a major role in Kalman filters and are abbreviated in this work as $N(\mu, \sigma^2)$ which specifies mean of the random variable and its variance. The normal distribution in equation 1.7 assumes that x is a scalar value. All Gaussian filters share the basic premise that beliefs are multivariate normal distributions. The PDF of a multivariate normal distribution is given by the following Gaussian function:

$$p(X) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp -\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu) \tag{1.8}$$

Where X is a multivariate vector, μ is the mean vector, Σ is a positive semidefinite and symmetric matrix called the covariance matrix.

The extended Kalman filter linearizes nonlinear Gaussian systems. Kalman filters implement belief computation for continuous states with all disturbances additive and Gaussian with zero mean.

$$X_t = a(U_t, X_{t-1}) + \varepsilon_t, \;\; \varepsilon_t \sim N(0, Q_t) \tag{1.9}$$

$$Z_t = b(X_t) + \delta_t, \;\; \delta_t \sim N(0, R_t) \tag{1.10}$$

Where a and b are nonlinear functions, U_t is the input, X_t is the state, Z_t is the observation, and $\varepsilon_t \sim N(0, Q_t)$ and $\delta_t \sim N(0, R_t)$ represent Gaussian random disturbances with zero mean and the specified covariance. a represents the state transition function and its purpose is to predict the current state $\overline{X}_t$, based on the previous state X_{t-1} and the current control input U_t. For a hypersonic vehicle, the control input to the extended Kalman filter could be the pilot's flight control commands (e.g., throttle, attitude controls, etc.) and sensor data (e.g., angle of attack, altitude, etc.). b represents the measurement transition function and are the expected measurements based on the current state X_t. ε_t is a Gaussian random variable that models the uncertainty introduced by the state transition function and δ_t describes the measurement noise.

Kalman filters assume a unimodal approximation to the true belief. A function is unimodal if for some value m (the mode), it is monotonically increasing for $x \leq m$ and monotonically decreasing for $x \geq m$. In that case, the maximum value of $f(x)$ is $f(m)$ and there are no other local maxima (Figure 1.7).

The Kalman filter assumes linear state and measurement models. Unfortunately, state transitions and measurements are rarely linear in practice. The extended Kalman filter relaxes this linear assumption by approximating the nonlinear state and measurement models with a first order

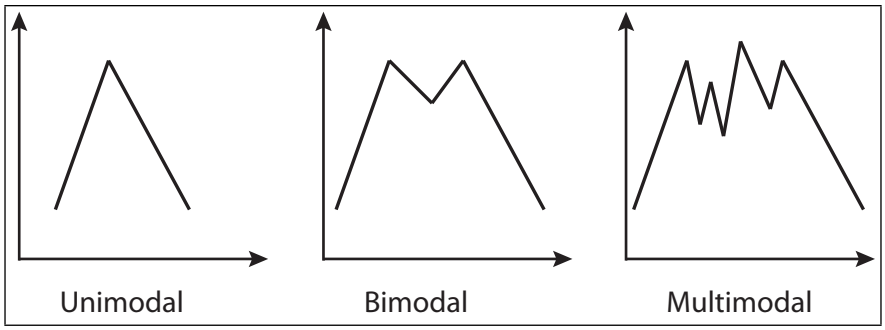

Figure 1.7 Unimodal illustration.

Taylor expansion linear model (Figure 1.8). Instead of passing the Gaussian through the nonlinear function g, it is passed through a linear approximation of g. The linear function is tangent to g at the mean of the original Gaussian. The resulting Gaussian is shown as the dashed line in the upper left graph. The linearization incurs an approximation error, as indicated by the mismatch between the linearized Gaussian (dashed) and the Gaussian computed from the highly accurate but expensive Monte-Carlo estimate (solid).

The extended Kalman filter is computationally quite efficient (Table 1.1). It is polynomial in measurement dimensionality k and state dimensionality $n : O(k^{2.4} + n^2)$ [Thrun et al., 2006]. The input to the extended Kalman filter is the belief at time $t-1$ represented by X_{t-1} and Σ_{t-1}. In step 1, the predicted state $\overline{X}_t$ is computed using the state transition function $a(U_t, X_{t-1})$ and the control input. The uncertainty estimate $\overline{\Sigma}_t$ grows in step 2 by incorporating the state model Jacobian A_t, the state model covariance Q_t, and the uncertainty from the previous time step Σ_{t-1}. The Kalman gain is computed in step 3 by leveraging the predicted covariance $\overline{\Sigma}_t$, the measurement transition Jacobian B_t which contain the derivatives of the measurements with respect to the state variables, and the measurement covariance matrix R_t. The Kalman gain specifies the degree that the measurement update Z_t is incorporated into the new state estimate X_t. The output is the belief at time t, represented by X_t and Σ_t, which are computed in steps 4 and 5 where the Kalman gain is incorporated. The measurement update corrects the predicted state $\overline{X}_t$ and shrinks the uncertainty. The filter represents the belief at time t by the state X_t and the covariance Σ_t.

1.4.2 Extended information filter

The extended information filter is also called the information form of the Kalman filter. The key difference between the extended Kalman filter and the extended information filter is the way the Gaussian belief is represented. In the extended Kalman filter, the Gaussians are represented

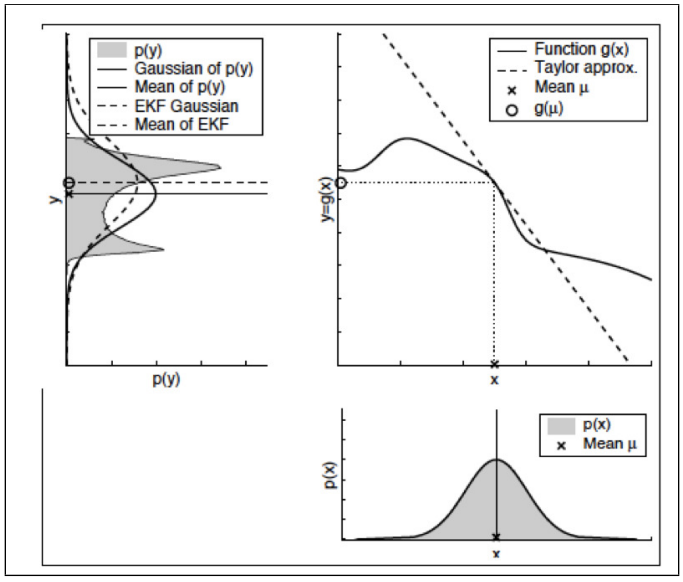

Figure 1.8 Illustration of linearization applied by the extended Kalman filter [Thrun et al., 2006].

Table 1.1 Extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t (Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t) \overline{\Sigma}_t$
6	Return to step 1 for next time step

Table 1.2 Extended information filter algorithm.

Step	Operation
1	$X_{t-1} = \Omega_{t-1}^{-1}\phi_{t-1}$
2	$\overline{\Omega}_t = (A\Omega_{t-1}^{-1}A^T + Q)^{-1}$
3	$\overline{\phi}_t = \overline{\Omega}_t a(U_t, X_{t-1})$
4	$\overline{X}_t = a(U_t, X_{t-1})$
5	$\Omega_t = \overline{\Omega}_t + B^T R^{-1} B$
6	$\phi_t = \overline{\phi}_t + BR^{-1}[Z_t - b(\overline{X}_t) + B\overline{X}_t]$
7	Return to step 1 for next time step

by their mean and covariance moments. In the extended information filter, the Gaussians are represented by their canonical parameterization where

$$\Omega = \Sigma^{-1} \qquad \phi = \Sigma^{-1}\mu \tag{1.11}$$

Ω is called the information matrix and ϕ is the information vector. The mean and covariance of the Gaussian can be obtained from the canonical parameterization by

$$\Sigma = \Omega^{-1} \qquad X = \Omega^{-1}\phi \tag{1.12}$$

The algorithm is presented in Table 1.2. U_t, $a(U_t, X_{t-1})$, A_t, $b(\overline{X}_t)$, B_t, Q_t, and R_t are identical to those in the extended Kalman filter. The prediction is implemented in steps 1 through 3 while the correction is implemented in steps 4 through 6.

The extended information filter is polynomial in measurement dimensionality k and state dimensionality $n : O(k^2 + n^{2.4})$. Comparison with the extended Kalman filter reveals the duality the these two filters. The measurement update is the difficult step in the extended Kalman filters because it requires a matrix inversion of a large matrix for every iteration. The extended information filter possesses an advantage of allowing R^{-1} to be computed once and reused for all iterations.

1.4.3 Particle filter

The particle filter is an alternative nonparametric implementation of the Bayes filter and is a Monte Carlo technique used for the solution of state estimation problems. The main idea is to represent the required posterior density function by a set of random samples with associated weights and to compute the estimates based on these samples and weights [Vianna et al., 2010].

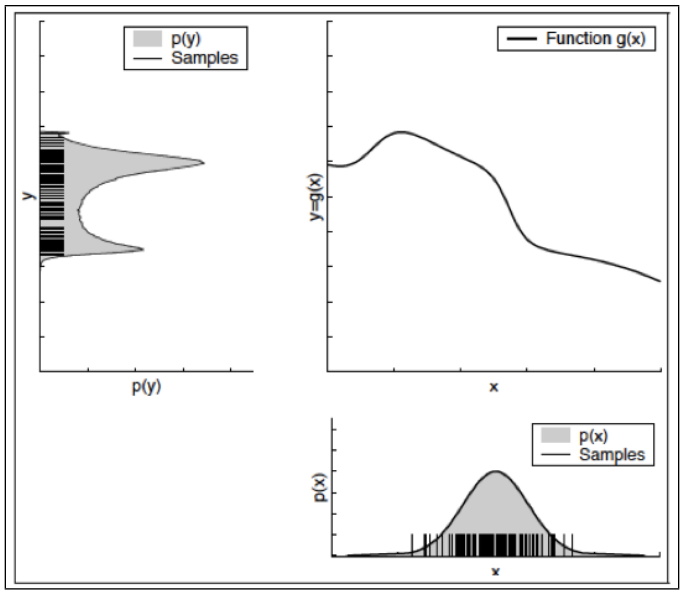

Figure 1.9 Illustration of how the particle filter represents the posterior density function [Thrun et al., 2006].

Figure 1.9 illustrates how particles are used to represent the belief. The lower right graph in Figure 1.9 shows samples drawn from a Gaussian random variable, x. The samples are passed through the nonlinear function $y = g(x)$ shown in the upper right graph. The resulting samples are distributed according to the random variable y which is the actual posterior density function. Figure 1.9 can be compared to the illustration of extended Kalman filter in Figure 1.8 where the posterior density function is approximated with a Gaussian. Because it is nonparametric, the particle filter can represent a much broader space of distributions than Gaussians and has the ability to model nonlinear transformations of random variables [Thrun et al., 2006]. The particle filter algorithm to locate the source can be found in Table 1.3.

1.4.4 Ordinary least squares

Ordinary least squares is applied to approximate solutions of overdetermined systems, i.e. systems of equations in which there are more equations than unknowns. Ordinary least squares is often applied in statistical contexts, particularly regression analysis. Ordinary least squares may be interpreted as a method of fitting data. The best fit, between modeled data and observed data, in its least-squares sense, is an instance of the model for which the sum of squared residuals has its least value, where a residual is the difference between an observed value and the value provided by the model. The method was first described by Carl Friedrich Gauss around 1794 [Bretscher, 1995].

Table 1.3 Particle filter algorithm.

Step	Operation
1	Generate m random possible source locations across the plate (particles)
2	Obtain expected measurements for each particle i for the current time t ($Z_{i,t}$)
3	Obtain actual measurements for the current time ($Ztrue_t$)
4	Weight each particle i according to $N(Ztrue_t, I)$
5	Normalize weights for each particle i into bins from 0 to 1
6	Resample best particles using normal distribution
7	Add position noise to each particle
8	Return to Step 2 for next time step

Ordinary least squares corresponds to the maximum likelihood criterion if the experimental errors have a normal distribution and can also be derived as a method of moments estimator [Woodbury, 2003a].

The ordinary least squares method is sometimes called the Gauss method of minimization [Woodbury, 2003a]. For a given state X, the value of T at $X = X + \Delta X$ is obtained through the truncated Taylor's series as

$$T|_{X+\Delta X} \approx T|_X + \left.\frac{\partial T}{\partial X}\right|_X \Delta X. \tag{1.13}$$

The ordinary least squares objective function is

$$S = (Y - T|_X - B\Delta X)^T (Y - T|_X - B\Delta X). \tag{1.14}$$

The minimizer of equation 1.14 is found by forcing to zero the derivative with respect to ΔX resulting in the estimator

$$\Delta X = (B^T B)^{-1} B^T (Y - T|_X). \tag{1.15}$$

The sensitivity matrix B can be normalized to produce a better conditioned matrix for the inversion in equation 1.15. As illustrated in Figure 1.10, X is updated using ΔX and a new ΔX is computed. Once each component in ΔX is small, the solution is converged yielding our estimate for the unknown parameters.

1.5 Goals and organization

The goal is for the reader to learn how to use ultrasound and the extended Kalman filter to locate concentrated heating sources. Development of the proposed method is accomplished using simple, controlled experiments involving concentrated high heat flux sources on a large flat

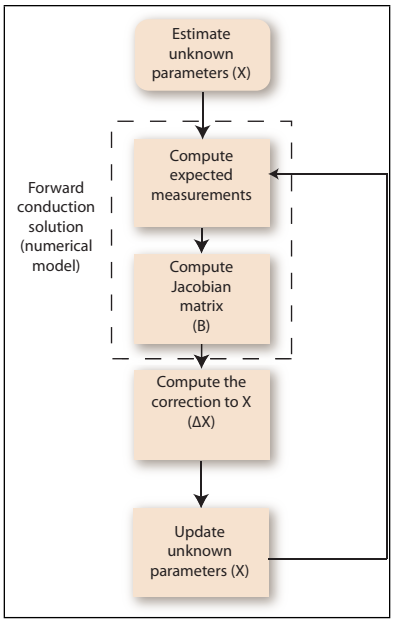

Figure 1.10 Ordinary least squares algorithm.

metal plate. Chapter II details a thermocouple and an ultrasound experiment with a spot source; a 3D forward conduction solution; six different measurement models for the inverse procedure; sensitivity to boundary conditions, thermal properties, and noise; and a comparison of parameter estimation methods. Chapter III details a thermocouple experiment with a step source; a 3D forward conduction solution; parameter estimation of experiment boundary conditions; sensitivity to boundary conditions, thermal properties, and noise; parameter estimation to locate both a static and a moving step heating source; and a sensor array design of experiments. Chapter IV shows how uncertainties in the extended Kalman filter can be addressed through an adaptive technique. The investigation encompasses sensitivity to changes in the state model covariance (Q), sensitivity to changes in the measurement covariance (R), the possibility that Q and R are correlated, behavior during convergence of the state covariance matrix (Σ), and behavior during convergence of the Kalman gain (K_t). Finally, the conclusions and future work are summarized in Chapter V.

CHAPTER II

SPOT SOURCE PARAMETER ESTIMATION

Development of the proposed parameter estimation method based on ultrasound and the extended Kalman filter is accomplished using simple, controlled experiments involving concentrated high heat flux sources on a large flat metal plate. A spot source is interesting for this problem because it presents a physical source. While a point source is mathematically interesting and closed-form solutions exist, a point source is non-physical and the underlying assumptions force solutions to diverge from actual conditions. This chapter details inverse method comparisons, flat plate experiments, measurement model selection, sensitivity to source position and boundary conditions, and heating source parameter estimation using a high heat flux spot source.

2.1 Inverse method selection

In order to provide justification for using the extended Kalman filter for the inverse procedure, a comparison is made in this section to quantify the performance of the extended Kalman filter against the performance of the ordinary least squares approach and the extended information filter. The extended Kalman filter and extended information filter are members of a family of recursive state estimators, collectively called Gaussian filters [Thrun et al., 2006]. The extended information filter is the information form of the Kalman filter. Both filters linearize nonlinear Gaussian systems. Ordinary least squares is sometimes called the Gauss method of minimization [Woodbury, 2003a] and is incredibly efficient and robust. In many situations, least squares outperforms all other optimization methods and therefore is commonly used as a benchmark. The inverse method selection is presented by first detailing a flat plate experiment, presenting a 3D conduction solution that matches the experiment, and then comparing parameter estimation results for determining the unknown boundary conditions using the extended Kalman filter, extended information filter, and ordinary least squares.

2.1.1 Flat plate experiment

Consider a 61 cm x 30.5 cm x 0.635 cm stainless steel 316L plate (Figure 2.1) with constant properties (Table 2.1). Four K-type thermocouples are attached on one side and four on the other. With plate center being the origin and the x-axis being the length (Figure 2.1), thermocouples were attached at (x,y) locations of (1 cm, 1 cm), (2 cm, 2 cm), (3 cm, 3 cm), and (-1 cm, -1 cm) on the heated side ($z = 0$) and on the non-heated side (z=0.635cm). The desire is to have thermocouple

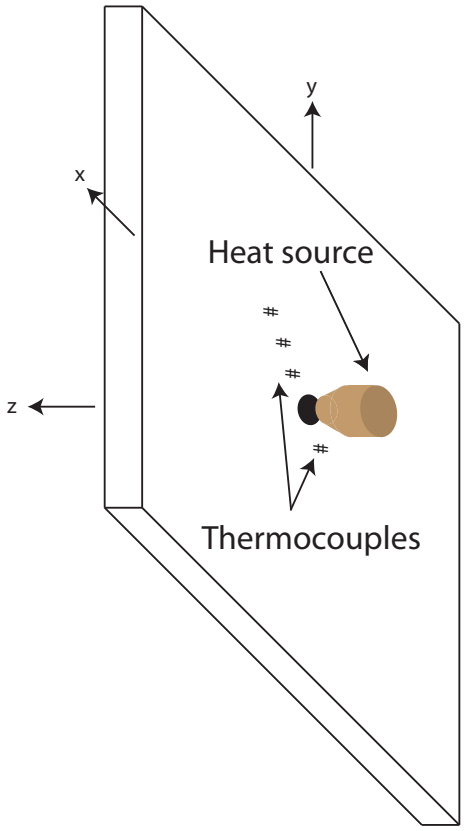

Figure 2.1 Illustration of flat plate with heat source and sensors (not drawn to scale).

pairs in exactly the same position on either side of the plate allowing measurement of the temperature difference between the two sides. The thermocouples are secured to the plate with thermal grease and Kapton tape to ensure good thermal contact. Flat black paint is applied to a 1.5 cm diameter area at the plate center to maximize energy absorption from the heater. The plate is oriented vertically with the positive y-axis pointing up. A Research, Inc. SpotIR® 4150 heater with focusing cone is positioned approximately 2 mm from the plate surface such that its beam strikes the plate center. Experiments are conducted with the heater running at full-power which, according to manufacturer's specifications, produces $1.7\,\mathrm{MW/m^2}$ of heat flux on the plate in a circular area 0.635 cm in diameter. Consequently, approximately 54 Watts of energy are being absorbed by the plate when the heater is on.

During the experiment, the heater is turned on at $t = 300\,\mathrm{s}$ and turned off and removed at $t = 600\,\mathrm{s}$. Data acquisition equipment is used to record thermocouple temperature readings every second during the experiment. A MIKRON Thermo Scan TS7302 infrared camera is used to collect thermal images of the plate and heater. Coupled with a laptop computer, this system records

Table 2.1 Material properties for the stainless steel 316L test sample used in the conduction experiments.

Property	Value
density (ρ)	$8,000\,\mathrm{kg/m^3}$
thermal conductivity (k)	$14.6\,\mathrm{W/m\,K}$
specific heat (c_p)	$500\,\mathrm{J/kg\,K}$
sound speed (v_0)	$5,100\,\mathrm{m/s}$ @ 293 K
ultrasonic TOF temperature factor (ξ)	$110 \times 10^{-6}\,1/\mathrm{K}$
sample length	61 cm
sample width	30.5 cm
sample height	0.635 cm

thermal images every five seconds during the experiment. Benefits gained from the thermal images include visualization of the temperature distribution throughout the experiment and the need to model secondary convection and radiation heating in addition to modeling the primary high heat flux coming from the heater's beam. Figure 2.2 illustrates the thermocouple temperature data recorded during the experiment. Analysis of the data indicates that a spatial temperature gradient of $6\,^{\circ}\mathrm{C/mm}$ exists during heating in the area of the thermocouple sets closest to the source [(1 cm, 1 cm) and (-1 cm, -1 cm)]. Positioning the heating source and the sensors within this degree of precision proved difficult. Therefore, sensor and heating source placement error is the most likely cause of the discrepancies between the two thermocouple sets closest to the source.

This experiment was conducted at Industrial Measurement Systems, Inc. in Aurora, IL.

2.1.2 3D conduction solution

The forward conduction solution leverages COMSOL Multiphysics® by the COMSOL Group and MATLAB® by The Mathworks, Inc. The COMSOL® model uses a finite element mesh with smaller elements near the heat source and larger elements near the plate edges to conserve computing resources.

For the flat plate detailed in Section 2.1.1, the governing equation for the subdomain (conduction in the plate) is

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = Q \tag{2.1}$$

where ∇ is the Laplacian and Q is an internal heat source (0 in this case). For the flat plate, k is

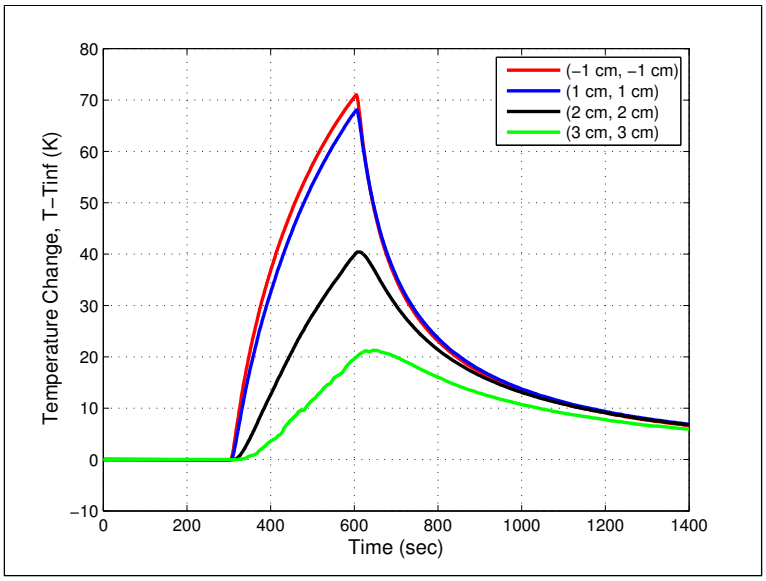

Figure 2.2 Temperature response on non-heated side of the plate at four sensor locations.

assumed constant. Thus, the subdomain governing equation is

$$\nabla^2 T = \frac{\rho C_p}{k}\frac{\partial T}{\partial t} \tag{2.2}$$

or

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \frac{\rho C_p}{k}\frac{\partial T}{\partial t} \tag{2.3}$$

The boundary condition is

$$n \cdot (k\nabla T) = q_0 + h(T_{inf} - T) \tag{2.4}$$

where n is the surface normal vector, q_0 is the inward heat flux. Radiation effects are assumed negligible for the plate.

The first meshing method analyzed in this section is the 3D free mesh using tetrahedral elements. A 0.635 cm diameter cylindrical subdomain in the plate's center is used to create a boundary for applying the heating source. This technique also creates small elements near the heating source and large elements far away from the source where temperature gradients are small thereby conserving computing resources. Mesh refinement is accomplished using all of the predefined free mesh sizes available in COMSOL® starting with the coarsest mesh and proceeding to the finest mesh. Grid convergence is achieved with 13,256 elements and 26,628 degrees of freedom, however the solution does not agree with an analytical solution of heating through a circular domain without convection [Kozlov et al., 1989] as illustrated in Figure 2.3. Element sizes from

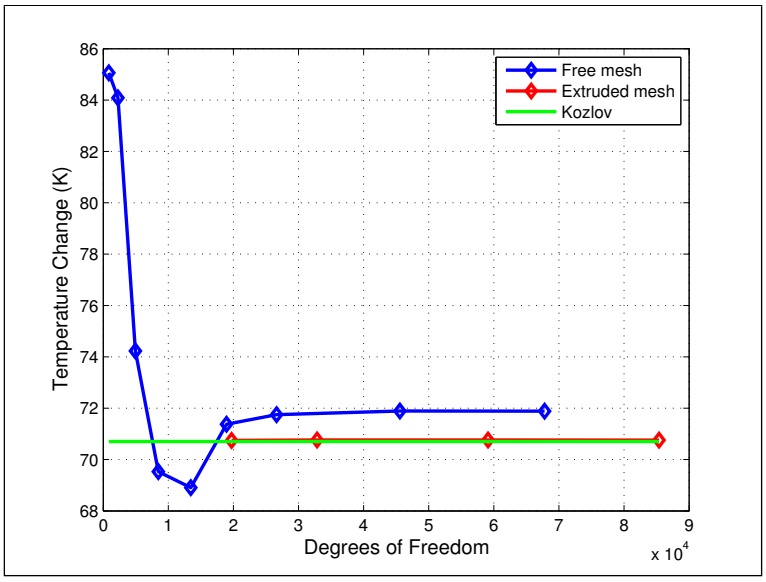

Figure 2.3 Grid independence results for the heated side at 1.4cm from the source and $t = 400\,\text{s}$.

the converged 3D free mesh were used to create an extruded mesh which does agree the analytic solution (Figure 2.3). The extruded mesh is generated by first creating 2D triangle elements in the plate's $x-y$ plane and then extruding the 2D mesh in the z-direction to create prism elements. Two subdomains consisting of a 0.635 cm diameter circle with a maximum element size of 1×10^3 m and a 6 cm diameter circle with a maximum element size of 5×10^{-3} m were used. The 2D mesh is created with the predefined normal free mesh setting in COMSOL®. The mesh extrusion process incorporates an option to create multiple mesh layers, therefore grid independence is contingent upon the number of layers through the thickness of the plate. The worst case is where the highest temperature gradients through the plate's thickness exist which is located at plate center. Figure 2.4 illustrates the computed temperature profile through the plate at plate center with one, two, four, and six mesh layers. The grid convergence study led to the selection of three mesh layers through the plate's thickness dimension, 9,780 total elements, and 45,983 degrees of freedom. Agreement between the final COMSOL® solution and the closed-form solution [Kozlov et al., 1989] is acceptable with mean absolute error less than 0.5 K.

2.1.3 Inverse methods comparison

Even with manufacturer specifications, the heat transfer between the radiative heater and the plate is not known with much certainty. Further complicating matters, the heater's proximity to the plate implies an unknown amount of secondary radiation and convection heating on the plate.

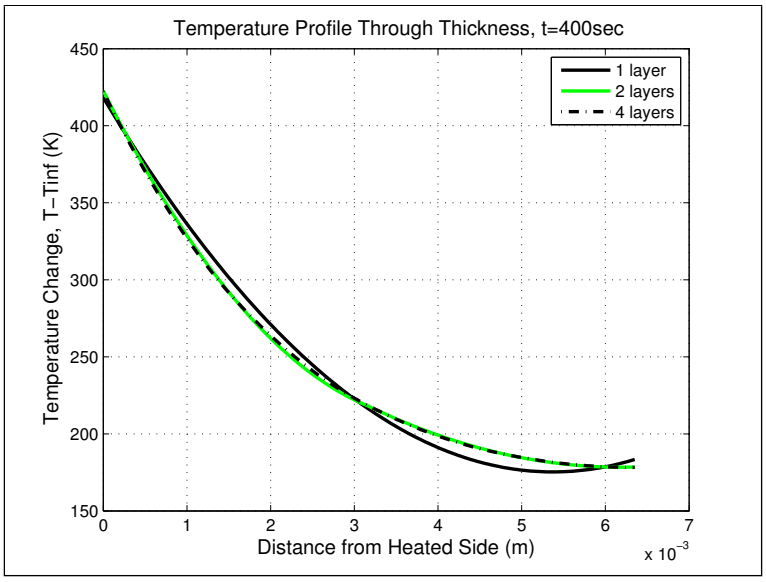

Figure 2.4 Number of mesh layers for best accuracy.

The focusing cone reaches temperatures in excess of 200 °C and the lamp is cooled with forced air that exits the heater through the focusing cone pointed at the plate. Based on the focusing cone temperature and a focusing cone area of 20 cm^2, approximately 2.5 W of radiation and convection energy are absorbed by the plate outside of area illuminated by the lamp. If we assume the area affected by this secondary heating is a circle with a radius of 9 cm, we can approximate the secondary heating with a Gaussian profile of $q''_g = 100\,\mathrm{W/m^2}$ and $\sigma_g^2 = 0.0009\,\mathrm{m^2}$. Figure 2.5 illustrates the boundary conditions used in modeling the plate. For this initial analysis, the main heat flux and convection coefficient are estimated. The convection coefficient being estimated is the average value over the duration of the experiment. While areas of the heated side of the plate may have a different convection coefficient value during heating, once the heater is removed, the average convection coefficient is identical on both sides of the plate. Thus, the heat transfer coefficient h is assumed constant and identical on both sides of the plate. Estimating h using free convection correlations [Incropera and DeWitt, 2002] produces an expected range of $2\,\mathrm{W/m^2K} \leq h \leq 5\,\mathrm{W/m^2K}$. Since the plate edges do not contribute significantly to the thermal load, $h = 3\,\mathrm{W/m^2K}$ is assumed on all four plate edges.

Three inverse methods are compared to quantify the heat flux (q'') and convection coefficient (h) on the plate: least squares, extended Kalman filter, and extended information filter. For the inversion, the entire experiment is treated as one event and temperature measurements are combined together. The experiment covers 1,400 seconds and data are recorded at one second

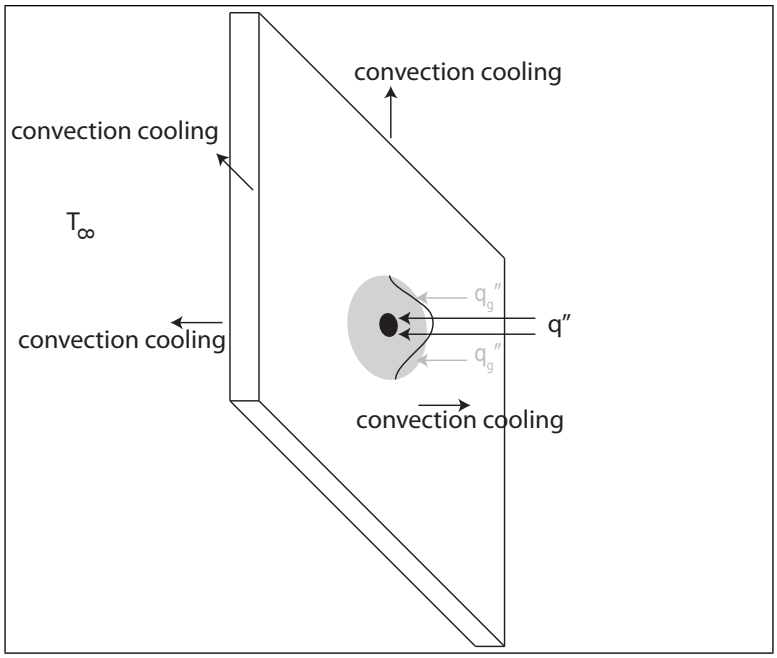

Figure 2.5 Illustration of boundary conditions on the flat plate.

intervals. Not all of the data is needed for the inverse and longer time steps can be used during periods of little thermal activity. Accordingly, one measurement at $t = 0\,\mathrm{s}$, one measurement per second from $t = 290$ to $800\,\mathrm{s}$ (the heater is on from $t = 300$ to $600\,\mathrm{s}$), and one measurement per five seconds from $t = 805$ to $1,400\,\mathrm{s}$ is used. The 5,056 temperature measurements therefore are effectively 5,056 separate sensors. All three methods start with an initial guess of the state $x_0 = [q''\ h]^{\mathrm{T}} = [1.7\,\mathrm{MW/m^2}\ 5.0\,\mathrm{W/m^2K}]^{\mathrm{T}}$ and are processed recursively to convergence.

For least squares, the estimated temperatures T for each sensor location and for each time t, a $5,056 \times 1$ matrix, depend on a vector of two unknowns in the state X and the value of T at $X = X + \Delta X$ is obtained through the truncated Taylor's series as [Woodbury, 2003b]

$$T|_{X+\Delta X} \approx T|_X + \left.\frac{\partial T}{\partial X}\right|_X \Delta X. \tag{2.5}$$

The gradient coefficient in equation 2.5 is the $5,056 \times 2$ sensitivity matrix

$$B = \frac{\partial T}{\partial X} = \begin{bmatrix} \frac{\partial T_1}{\partial q''} q'' & \frac{\partial T_1}{\partial h} h \\ \frac{\partial T_2}{\partial q''} q'' & \frac{\partial T_2}{\partial h} h \\ \vdots & \vdots \\ \frac{\partial T_{5,056}}{\partial q''} q'' & \frac{\partial T_{5,056}}{\partial h} h \end{bmatrix} \tag{2.6}$$

which has been normalized to have units of temperature producing a better conditioned matrix for the inversion in the least squares estimator $\Delta X = (B^T B)^{-1} B^T (Y - T|_X)$ [Woodbury, 2003b].

Our COMSOL® numerical model produces values of T based on current estimates for the state X. The sensitivity matrix B is obtained using finite differences (a $5,056 \times 2$ matrix) by independently varying the state parameters 0.1%. We designate our experimentally obtained temperature measurements as Y, a $5,056 \times 1$ matrix. Our goal is to improve the estimate for the state X based on the observations Y.

The algorithm for the extended Kalman filter is listed in Table 2.2 where $\overline{X}_t$ is the predicted state, $a(U_t, X_{t-1})$ is the state model based on the input U_t and the previous state X_{t-1}, A is the state model Jacobian, $\overline{\Sigma}$ is the uncertainty estimate, Q_t is the state model covariance, K_t is the Kalman gain, B_t is the measurement Jacobian, R_t is the measurement covariance, $b(\overline{X}_t)$ is the measurement transition function and represents the predicted measurements from the forward conduction solution based on the predicted state, and Z_t represents the actual measurements. The filter represents the belief at time t by the state X_t and the covariance Σ_t. For the flat plate considered here, there is no input to the state thus the state model is $a = I_2$ and the state model Jacobian is $A = I_2$, where I_2 is a 2×2 identity matrix. The measurement transition function b is a $5,056 \times 1$ matrix of the predicted temperatures from the forward conduction solution, and the measurement Jacobian B is obtained using finite differences (a $5,056 \times 2$ matrix) by independently varying the state parameters 0.1%. The state model covariance matrix Q is a 2×2 diagonal matrix using $\sigma_q^2 = 0.1\,\mathrm{MW}^2/\mathrm{m}^4$ and $\sigma_h^2 = 0.1\,\mathrm{W}^2/\mathrm{m}^4\mathrm{K}^2$. These values were chosen through a parameter sweep to achieve smooth convergence behavior since small values for the state model covariance matrix cause the Gaussian filters to diverge while arbitrarily large values for the state model covariance matrix render the Gaussian filters essentially identical to the least squares method. The thermocouples have a measurement accuracy of $\pm 1.5\,^\circ\mathrm{C}$, which translates to a measurement variance of $\sigma_T^2 = 0.25\,^\circ\mathrm{C}^2$. This value is used for the diagonal elements of the measurement covariance matrix R, a $5,056 \times 5,056$ matrix. The filter is initialized with the initial state x_0 (stated above) and covariance $\Sigma_0 = 0$. For the extended information filter (Table 2.3), a, A, b, B, R, and Q are identical to those in the extended Kalman filter. The extended information filter possesses an advantage of allowing the inverse of the measurement covariance matrix Q^{-1} to be computed once and reused for all iterations. Because the initial state covariance matrix Σ_0 is inverted in the extended information filter, the filter is initialized with $\Sigma_0 = R$ instead of the zero matrix used to initialize the extended Kalman filter.

Figures 2.6 and 2.7 illustrate the convergence behavior for all three methods. The extended Kalman filter and extended information filter converge identically and are presented together. The Gaussian filters converge a bit slower than the least squares method, however the convergence is smoother. Once convergence is achieved, statistical moments are computed from the last three iterations (Figure 2.8). Results are similar for all three methods. The least squares method uses

Table 2.2 Extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t (Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t) \overline{\Sigma}_t$
6	Return to step 1 if solution not converged

Table 2.3 Extended information filter algorithm.

Step	Operation
1	$X_{t-1} = \Omega_{t-1}^{-1} \phi_{t-1}$
2	$\overline{\Omega}_t = (A \Omega_{t-1}^{-1} A^T + Q)^{-1}$
3	$\overline{\phi}_t = \overline{\Omega}_t a(U_t, X_{t-1})$
4	$\overline{X}_t = a(U_t, X_{t-1})$
5	$\Omega_t = \overline{\Omega}_t + B^T R^{-1} B$
6	$\phi_t = \overline{\phi}_t + B R^{-1} [Z_t - b(\overline{X}_t) + B \overline{X}_t]$
7	Return to step 1 if solution not converged

the least amount of wall time and memory of the three methods. Wall time for each iteration, independent of recomputing the COMSOL® model for the updated parameters, is approximately two orders of magnitude longer for extended information filter and four orders of magnitude longer for extended Kalman filter than the wall time for least squares. Memory usage is approximately 1.5 times more for extended information filter and 2 times more for extended Kalman filter than the memory required by least squares. When considering convergence behavior and computational cost, least squares outperforms the other methods for this type of parameter estimation.

Figure 2.9 compares the temperature response measured during the experiment with the temperature response of the model using the results of the estimation (i.e., $q'' = 0.930\,\mathrm{MW/m^2}$ and $h = 3.20\,\mathrm{W/m^2\,K}$). The residuals [Beck and Woodbury, 1998, Dowding and Blackwell, 2001] are illustrated in Figure 2.10. Agreement between the model and the experiment is acceptable, however improvement could be achieved through modifications to the heating profile (e.g., secondary heating). Agreement with the experiment is better when simultaneously estimating q'' and h than when estimating q'' with h arbitrarily fixed.

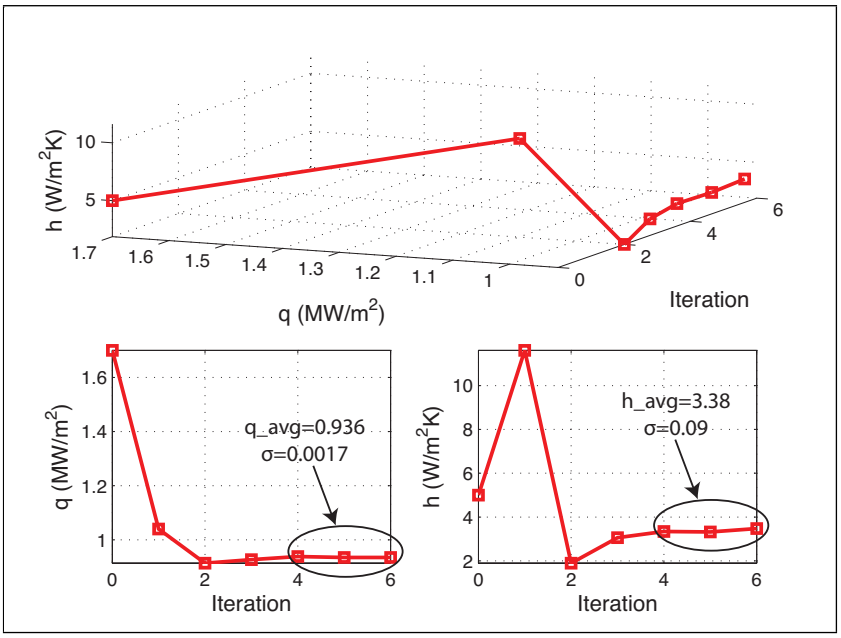

Figure 2.6 Least squares convergence.

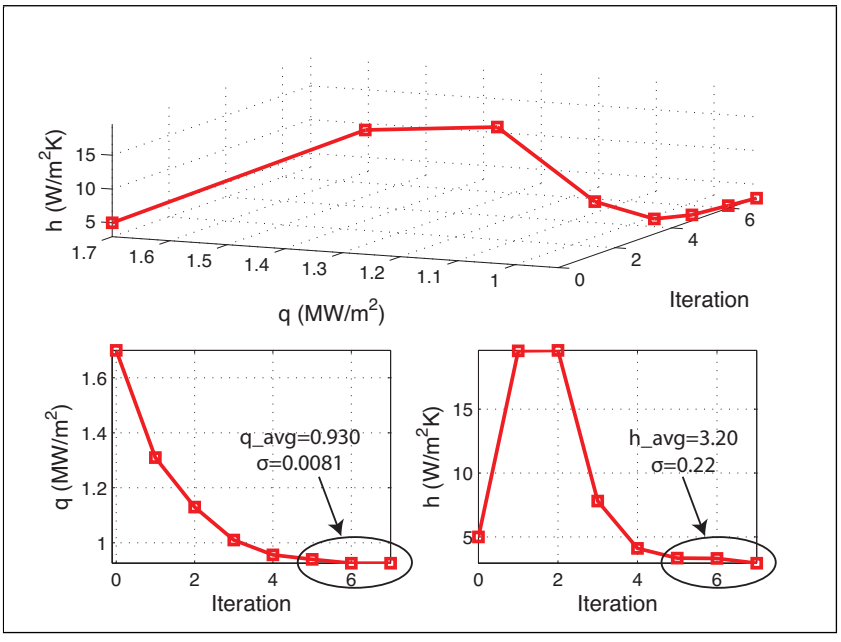

Figure 2.7 Extended Kalman filter and extended information filter convergence. The filters produce identical results and are presented together.

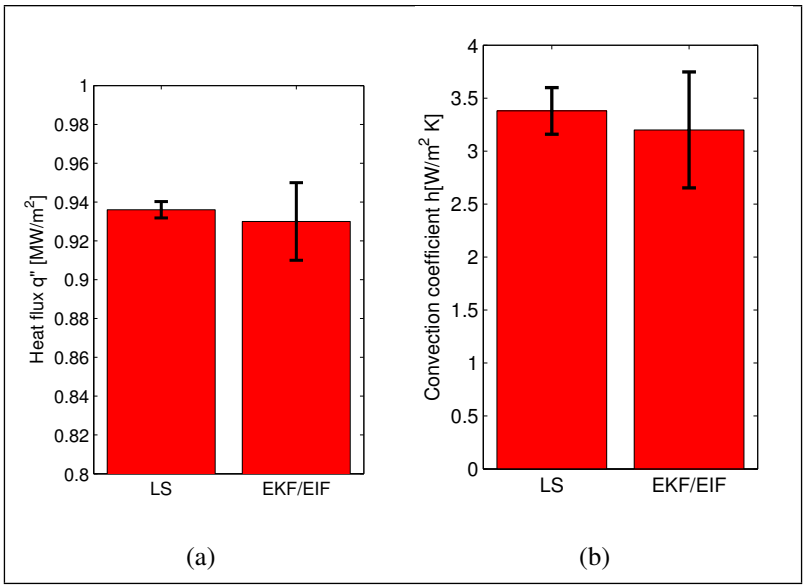

Figure 2.8 Statistical moments from parameter identification for (a) heat flux q'' and (b) convection coefficient h comparing least squares, extended Kalman filter, and extended information filter.

A check of the boundary effect errors is conducted to ensure the plate is sized sufficiently large (Figure 2.11). Of particular interest is in the region of ($\pm$4 cm,$\pm$4 cm) where the errors remain well below 0.5% for the entire experiment. Even at ($\pm$10 cm,$\pm$10 cm), the errors are below 1% for much of the experiment and stay below 3% for the entire experiment.

2.2 Measurement model selection

One important question in this analysis is determining if an ultrasound-based solution can outperform a thermocouple solution. This section seeks to answer this question by examining six measurement models, two incorporating thermocouples and four incorporating ultrasonic transducers. The following six measurement models have been identified for analysis and will be detailed in the following sections:

1. Temperature measurement model
2. Radius from temperature measurement model
3. Ultrasonic pulse-echo time of flight measurement model
4. Radius from ultrasonic pulse-echo time of flight measurement model
5. Ultrasonic pulse one-way time of flight measurement model

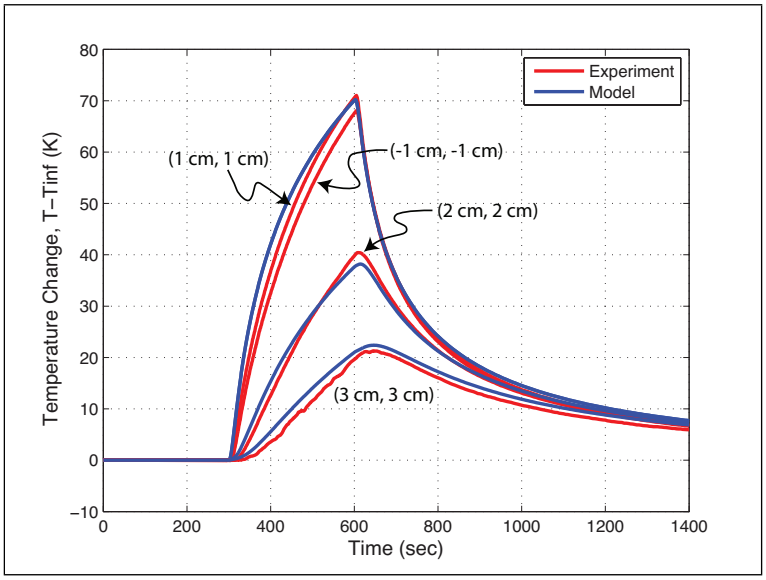

Figure 2.9 Comparison of the temperature response on non-heated side of the plate at four sensor locations.

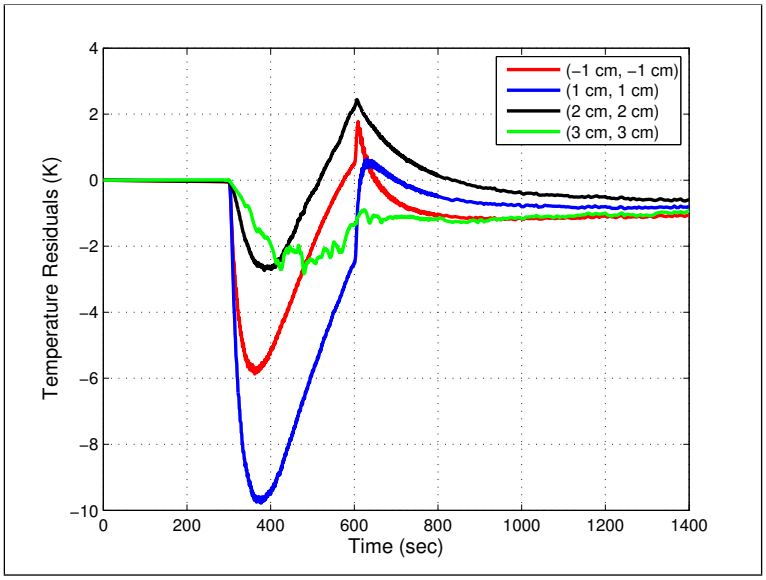

Figure 2.10 Residuals of the model when compared to the experiment measurements on the non-heated side.

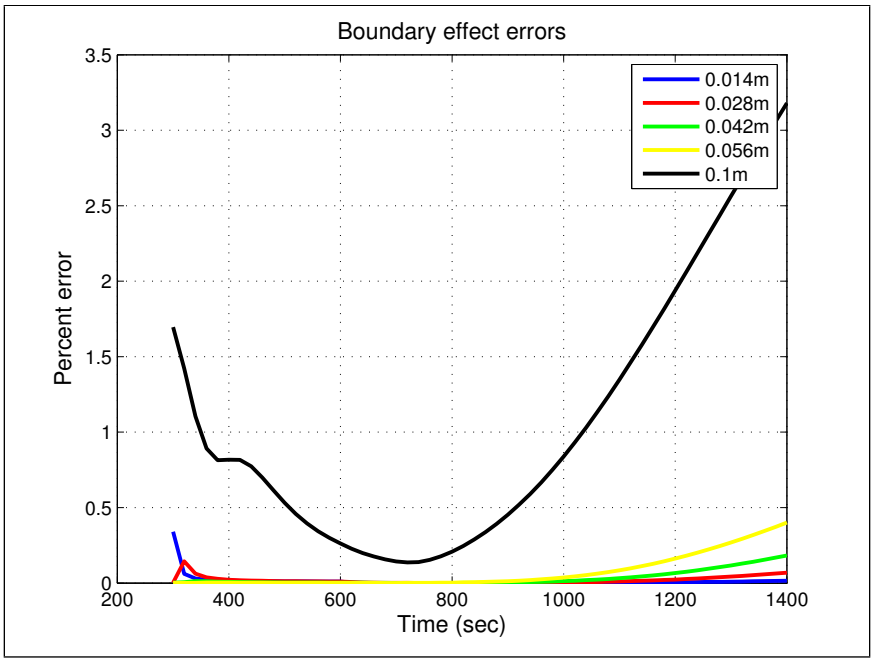

Figure 2.11 Illustration of when and where the plate edges introduce errors in the temperature distribution.

6. Ellipse from ultrasonic one-way pulse time of flight measurement model

These measurement models represent different ways to collect measurements (sensors) and different ways to process the data. Comparison of the six measurement models is performed using the extended Kalman filter (algorithm in Table 2.2) to locate the source (x_q, y_q). Figure 2.12 illustrates the parameter estimate update process for the extended Kalman filter and the COMSOL® model. For all six measurement models, the estimated state is $X_t = [x_s, y_s]^T$ where x_s and y_s represent the estimated location of the source at time t. There is no input to the state thus the state model is $a = I_2$ and the state model Jacobian is $A = I_2$. Sensitivity of the state variance is compared for values from $\sigma^2 = 0.01\,\mathrm{m}^2$ to $0.000001\,\mathrm{m}^2$ with the lower values providing a damping effect. A state variance of $\sigma^2 = 0.0001\,\mathrm{m}^2$ provides smooth, fast convergence without producing erratic convergence behavior exhibited by the higher state variance values and will be used for all measurement model comparisons in this section. Thus, the state model covariance matrix is $Q_t = 0.0001\,\mathrm{m}^2 * I_2$, where I_2 is a 2×2 identity matrix.

Locating and characterizing a heating source depends upon many factors such as heating source movements in time, heating source magnitude changes in time, and other transient behaviors. Fairly restrictive assumptions can be imposed that simplify the problem. Analysis and algorithm development can proceed using these restrictive assumptions and then assumptions can

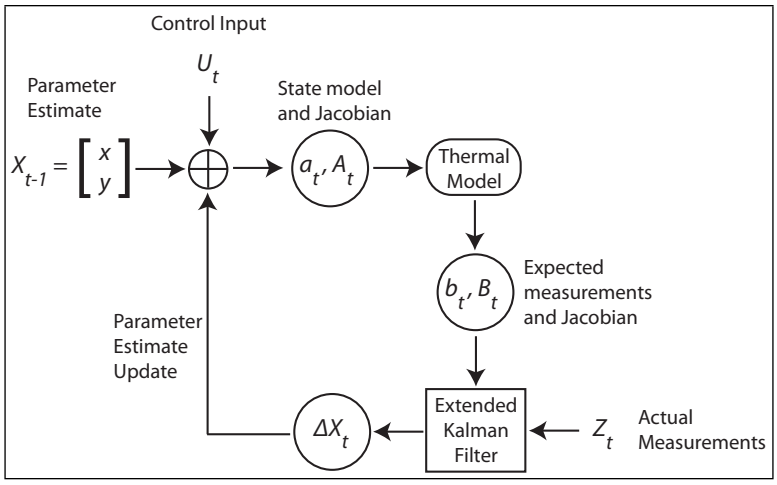

Figure 2.12 Parameter estimate update process for the extended Kalman filter and the COMSOL® model.

be relaxed in stages to achieve the end result of source localization and characterization. The assumptions for this section are:

1. Source in fixed position (location unknown)
2. Source applied at time $t = 300\,\text{s}$ and removed at $t = 600\,\text{s}$
3. $q'' = 0.930\,\text{MW}/\text{m}^2$ over $0.00635\,\text{m}$ diameter circular area while source applied (value obtained in parameter estimation above)
4. Secondary heating is characterized by a Gaussian with magnitude $q''_g = 100\,\text{W}/\text{m}^2$ and variance $\sigma_g^2 = 0.0009\,\text{m}^2$ while source applied
5. Convection coefficient $h = 3.20\,\text{W}/\text{m}^2\text{K}$ on both sides of the plate (value obtained in parameter estimation above)
6. Convection coefficient $h = 3\,\text{W}/\text{m}^2\text{K}$ on the plate edges
7. Thermal conductivity $k = 15\,\text{W}/\text{mK}$
8. Specific heat $C_p = 500\,\text{J}/\text{kgK}$ and density $\rho = 8,000\,\text{kg}/\text{m}^3$
9. Positions of sensors are ($\pm 4\,\text{cm}$, $\pm 4\,\text{cm}$) on the non-heated side

2.2.1 Temperature measurement model

In this direct model, temperatures are measured using four thermocouples on the non-heated side of the plate. Expected temperatures and the partial derivatives are obtained directly from COMSOL® to form the measurement transition function $b(\overline{X}_t)$ and the Jacobian B_t.

$$b(\overline{X}_t) = \begin{bmatrix} \overline{\theta}_1 \\ \overline{\theta}_2 \\ \overline{\theta}_3 \\ \overline{\theta}_4 \end{bmatrix} \tag{2.7}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{\theta}_1}{\partial x_1} & -\frac{\partial \overline{\theta}_1}{\partial y_1} \\ -\frac{\partial \overline{\theta}_2}{\partial x_2} & -\frac{\partial \overline{\theta}_2}{\partial y_2} \\ -\frac{\partial \overline{\theta}_3}{\partial x_3} & -\frac{\partial \overline{\theta}_3}{\partial y_3} \\ -\frac{\partial \overline{\theta}_4}{\partial x_4} & -\frac{\partial \overline{\theta}_4}{\partial y_4} \end{bmatrix} \tag{2.8}$$

where t is time in seconds with a time step of 1 s, $\overline{\theta}$ is the expected change in temperature relative to a reference, obtained from COMSOL®, if the heating source is located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ indicating the locations of the four thermocouples. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience since this information can be obtained with one COMSOL® simulation. The derivatives are obtained directly from COMSOL®. Based on the flat plate experiment above, sensor noise is assumed be $\pm 0.045\,\mathrm{K}$ and is normally distributed ($\sigma^2 = (0.045/3)^2 = 2.225 \times 10^{-4}\,\mathrm{K}^2$). The measurement covariance matrix is $R = 2.225 \times 10^{-4}\,\mathrm{K}^2 * I_4$.

2.2.2 Radius from temperature measurement model

This indirect model is similar to the previous model in that temperatures are measured using thermocouples, but in this model, COMSOL® is used as a lookup table to convert measured temperatures to a radius from each sensor to the source. Knowledge of the heating start time, one of the assumptions in this section, enables a simple COMSOL® lookup of expected temperatures for a range of radius values from the heating source. Linear interpolation is used with this lookup table to obtain an expected radius for each temperature measurement.

$$r_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{2.9}$$

where (x_i, y_i) is the location of sensor i for $i = 1,2,3,4$ and (x_s, y_s) the heating source location. The Jacobian is based solely on geometry, which may reduce errors.

$$\frac{\partial r_i}{\partial x_s} = \frac{1}{2}\left((x_i - x_s)^2 + (y_i - y_s)^2\right)^{-\frac{1}{2}} \left(\frac{\partial}{\partial x_s}(x_i^2 - 2x_i x_s + x_s^2)\right) \tag{2.10}$$

$$\frac{\partial r_i}{\partial x_s} = \frac{x_s - x_i}{r_i}, \frac{\partial r_i}{\partial y_s} = \frac{y_s - y_i}{r_i} \tag{2.11}$$

The measurement transition function $b(\overline{X}_t)$ and the Jacobian B_t are then

$$b(\overline{X}_t) = \begin{bmatrix} \sqrt{(x_1 - x_s)^2 + (y_1 - y_s)^2} \\ \sqrt{(x_2 - x_s)^2 + (y_2 - y_s)^2} \\ \sqrt{(x_3 - x_s)^2 + (y_3 - y_s)^2} \\ \sqrt{(x_4 - x_s)^2 + (y_4 - y_s)^2} \end{bmatrix} \tag{2.12}$$

$$B_t = \begin{bmatrix} \frac{x_s - x_1}{r_1} & \frac{y_s - y_1}{r_1} \\ \frac{x_s - x_2}{r_2} & \frac{y_s - y_2}{r_2} \\ \frac{x_s - x_3}{r_3} & \frac{y_s - y_3}{r_3} \\ \frac{x_s - x_4}{r_4} & \frac{y_s - y_4}{r_4} \end{bmatrix} \tag{2.13}$$

where t is time in seconds with a time step of 1 s, $\overline{r}_i$ with $i = 1,2,3,4$ is the radius from the sensor to the source, obtained from COMSOL®, if the source is located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ indicating the locations of the four thermocouples. Based on the flat plate experiment above, sensor noise is assumed be ± 0.045 K and is normally distributed ($\sigma^2 = (0.045/3)^2 = 0.000225\,\mathrm{K}^2$). Since measured temperature is being related to radius, sensor noise must be converted into radius noise. The complication in this conversion arises from the fact that radius is a non-linear function of temperature and time. Based on insights gained from the forward conduction model and analysis of the temperature response in the plate, a value of $0.015\,\mathrm{m/K}$ is used resulting in a radius noise of ± 0.000675 m with normal distribution ($\sigma^2 = 5.06 \times 10^{-8}\,\mathrm{m}^2$). The measurement covariance matrix, therefore, is $R = 5.06 \times 10^{-8}\,\mathrm{m}^2 * I_4$.

2.2.3 Ultrasonic pulse-echo time of flight measurement model

This direct model uses ultrasonic pulses to measure the average temperature through the material thickness at each sensor location. In the pulse-echo method, the ultrasonic pulse travels through the material thickness, reflects off the boundary, and returns to the transducer. The time of

flight is [Myers et al., 2008, Myers et al., 2010c]

$$G_{ii} = \frac{2L}{v_o}\left(1 + \xi\,\theta_{avg}\Big|_0^L\right) \tag{2.14}$$

where L represents the material thickness, v_0 is the speed of sound in the material at a reference temperature, ξ is the ultrasonic time of flight factor which is material dependent, and θ is the change in temperature from the reference temperature. The ultrasonic pulse time of flight measurement model consists of obtaining expected temperatures from COMSOL®, computing the average temperature between the transducer and the boundary, and then computing an expected time of flight using equation 2.14 to form the measurement transition function $b(\overline{X}_t)$ (equation 2.15). The Jacobian partial derivatives are obtained using time of flight difference when moving the source in the x and y directions independently (equation 2.16).

$$b(\overline{X}_t) = \begin{bmatrix} \overline{G}_1 \\ \overline{G}_2 \\ \overline{G}_3 \\ \overline{G}_4 \end{bmatrix} \tag{2.15}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{G}_1}{\partial x_1} & -\frac{\partial \overline{G}_1}{\partial y_1} \\ -\frac{\partial \overline{G}_2}{\partial x_2} & -\frac{\partial \overline{G}_2}{\partial y_2} \\ -\frac{\partial \overline{G}_3}{\partial x_3} & -\frac{\partial \overline{G}_3}{\partial y_3} \\ -\frac{\partial \overline{G}_4}{\partial x_4} & -\frac{\partial \overline{G}_4}{\partial y_4} \end{bmatrix} \tag{2.16}$$

where t is time in seconds with a time step of 1 second, $\overline{G}_i$ with $i = 1,2,3,4$ is the expected ultrasonic pulse time of flight, obtained from COMSOL®, with the heating source at location (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ indicating the locations of the four transducers. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience since this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the x and y positions of all sensors by 0.0001 m. Based on the flat plate experiment above, sensor noise is assumed be $\pm 2.3 \times 10^{-10}$ s and is normally distributed ($\sigma^2 = 5.88 \times 10^{-21}\,\text{sec}^2$). The measurement covariance matrix, therefore, is $R = 5.88 \times 10^{-21}\,\text{sec}^2 * I_4$.

2.2.4 Radius from ultrasonic pulse-echo time of flight measurement model

In this indirect model, ultrasonic pulse-echo time of flight is measured using four transducers on the non-heated side of the plate. Similar to radius from temperature method, this method converts the measured time of flight to a radius using the COMSOL® model as a lookup table. Knowledge of the heating start time, one of the assumptions in this section, enables a simple COMSOL® lookup of expected temperatures for a range of radius values from the heating source. Temperatures in the plate are related to time of flight through equation 2.14. Linear interpolation is used with this lookup table to obtain an expected radius for each time of flight measurement. Equations 2.9 to 2.11 develop the geometry behind the measurement transition function $b(\overline{X}_t)$ and the Jacobian B_t which are

$$b(\overline{X}_t) = \begin{bmatrix} \sqrt{(x_1 - x_s)^2 + (y_1 - y_s)^2} \\ \sqrt{(x_2 - x_s)^2 + (y_2 - y_s)^2} \\ \sqrt{(x_3 - x_s)^2 + (y_3 - y_s)^2} \\ \sqrt{(x_4 - x_s)^2 + (y_4 - y_s)^2} \end{bmatrix} \tag{2.17}$$

$$B_t = \begin{bmatrix} \frac{x_s - x_1}{r_1} & \frac{y_s - y_1}{r_1} \\ \frac{x_s - x_2}{r_2} & \frac{y_s - y_2}{r_2} \\ \frac{x_s - x_3}{r_3} & \frac{y_s - y_3}{r_3} \\ \frac{x_s - x_4}{r_4} & \frac{y_s - y_4}{r_4} \end{bmatrix} \tag{2.18}$$

where t is time in seconds with a time step of 1 second, $\bar{r}_i$ with $i = 1,2,3,4$ is the radius from the sensor to the source, obtained from COMSOL®, if the source is located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ indicating the locations of the four thermocouples. Based on the flat plate experiment above, sensor noise is assumed be $\pm 2.3 \times 10^{-10}$ s and is normally distributed ($\sigma^2 = 5.88 \times 10^{-21}\,\text{sec}^2$). Sensor noise in terms of temperature can be expressed as

$$\theta_{noise} = \frac{G_{noise} v_0}{2L\xi} = 0.84\,\text{K} \tag{2.19}$$

Since measured time of flight is being related to radius, sensor noise must be converted into radius noise. The complication in this conversion arises from the fact that radius is a non-linear function of time of flight and time. Based on insights gained from the forward conduction model and analysis of the temperature response in the plate, a value of $0.015\,\text{m/K}$ is used resulting in a radius noise of $\pm 0.0126\,\text{m}$ with normal distribution ($\sigma^2 = 1.76 \times 10^{-5}\,\text{m}^2$). The measurement covariance matrix, therefore, is $R_t = 1.76 \times 10^{-5}\,\text{m}^2 * I_4$.

2.2.5 Ultrasonic pulse one-way time of flight measurement model

Instead of sending an ultrasonic pulse through to a boundary and receiving the echo at the original transducer, one transducer can transmit the pulse and another transducer can receive the pulse. The time of flight is

$$G_{ij} = \frac{R_{ij}}{v_o}\left(1 + \xi\,\theta_{avg}\big|_i^j\right) \tag{2.20}$$

where R_{ij} is the distance between transducers (m). This direct measurement model consists of obtaining expected temperatures from COMSOL®, computing the average temperature between the transducers, and then computing an expected time of flight to form $a(U_t, X_{t-1})$ (equation 2.21). For the current analysis, the average temperature is based on the line on the plate surface between the two sensors. The Jacobian partial derivatives are obtained using time of flight difference when moving the source in the x and y directions independently (equation 2.22).

$$b(\overline{X}_t) = \begin{bmatrix} \overline{G}_1 \\ \overline{G}_2 \\ \overline{G}_3 \\ \overline{G}_4 \end{bmatrix} \tag{2.21}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{G}_1}{\partial x_1} & -\frac{\partial \overline{G}_1}{\partial y_1} \\ -\frac{\partial \overline{G}_2}{\partial x_2} & -\frac{\partial \overline{G}_2}{\partial y_2} \\ -\frac{\partial \overline{G}_3}{\partial x_3} & -\frac{\partial \overline{G}_3}{\partial y_3} \\ -\frac{\partial \overline{G}_4}{\partial x_4} & -\frac{\partial \overline{G}_4}{\partial y_4} \end{bmatrix} \tag{2.22}$$

where t is time in seconds with a time step of 1 s, $\overline{G}_i$ with $i = 1,2,3,4$ is the ultrasonic pulse time of flight, obtained from COMSOL®, with the heating source located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ indicating the locations of the four transducers. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience since this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the x and y positions of all sensors by 0.0001 m. Based on the flat plate experiment above, sensor noise is assumed be $\pm 1.05 \times 10^{-8}$ s and is normally distributed ($\sigma^2 = ((1.05 \times 10^{-8})/3)^2 = 1.225 \times 10^{-17}\,\text{sec}^2$). The measurement covariance matrix, therefore, is $R = 1.225 \times 10^{-17}\,\text{sec}^2 * I_4$.

2.2.6 Ellipse from ultrasonic pulse one-way time of flight measurement model

In this indirect model, a particular ultrasonic pulse time of flight at a particular time after the heater is turned on means that the source could be anywhere on an assumed elliptical shape around

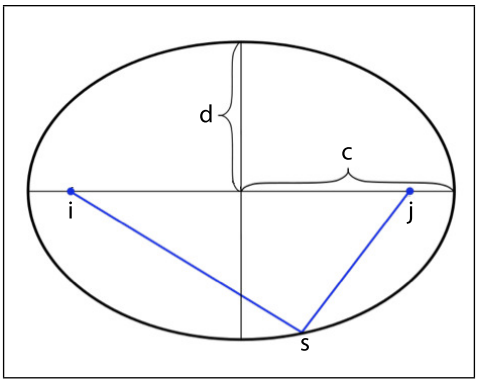

Figure 2.13 Ellipse properties.

the sensors. Figure 2.13 illustrates the geometry of an ellipse. The two sensors are assumed to be the focus points for the ellipse. Since the distance between sensors is known, ellipse parameters c and d can be related to each other and the ellipse can be represented with just one parameter c.

$$r_{is} + r_{js} = 2c = \sqrt{r_{ij}^2 + 4d^2} \tag{2.23}$$

where i and j are sensors and s is heat source.

$$c = \frac{1}{2}\sqrt{r_{ij}^2 + 4d^2} = \frac{r_{is} + r_{js}}{2} \tag{2.24}$$

$$r_{is} = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{2.25}$$

$$r_{js} = \sqrt{(x_j - x_s)^2 + (y_j - y_s)^2} \tag{2.26}$$

$$\frac{\partial c_i}{\partial x_s} = \frac{1}{2}\left[\frac{x_s - x_i}{r_{is}} + \frac{x_s - x_j}{r_{js}}\right] \tag{2.27}$$

$$\frac{\partial c_i}{\partial y_s} = \frac{1}{2}\left[\frac{y_s - y_i}{r_{is}} + \frac{y_s - y_j}{r_{js}}\right] \tag{2.28}$$

The parameter c is measured indirectly by first measuring the one-way ultrasonic pulse time of flight. The forward conduction solution is used to get time of flight for a range of c values and interpolated using the spline method to obtain c for the measured time of flight. The measurement

transition function $b(\overline{X}_t)$ and the Jacobian B_t are then

$$b(\overline{X}_t) = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \tag{2.29}$$

$$B_t = \begin{bmatrix} \frac{\partial c_1}{\partial x_s} & \frac{\partial c_1}{\partial y_s} \\ \frac{\partial c_2}{\partial x_s} & \frac{\partial c_2}{\partial y_s} \\ \frac{\partial c_3}{\partial x_s} & \frac{\partial c_3}{\partial y_s} \\ \frac{\partial c_4}{\partial x_s} & \frac{\partial c_4}{\partial y_s} \end{bmatrix} \tag{2.30}$$

where t is time in seconds with a time step of 1 s, c_i with $i = 1,2,3,4$ is the ellipse parameter if the source is located at (x_s, y_s). Based on the flat plate experiment above, sensor noise is assumed be $\pm 1.05 \times 10^{-8}$ s and is normally distributed ($\sigma^2 = 1.22 \times 10^{-17}\,\mathrm{s}^2$). The sensor noise in terms of temperature can be expressed as

$$\theta_{noise} = \frac{G_{noise} v_0}{L\xi} = 6.09\,\mathrm{K} \tag{2.31}$$

Since measured time of flight is being related to the ellipse parameter c, sensor noise must be converted into ellipse parameter noise. The complication in this conversion arises from the fact that c is a non-linear function of time of flight and time. Based on insights gained from the forward conduction model and analysis of the temperature response in the plate, a value of $0.015\,\mathrm{m/K}$ is used resulting in an ellipse noise of $\pm 2.04 \times 10^4$ m for the c parameter with normal distribution ($\sigma^2 = 4.62 \times 10^{-9}\,\mathrm{m}^2$).

2.2.7 Extended Kalman filter convergence behavior

Extended Kalman filter convergence behavior for all six measurement models are compared in Figures 2.14 through 2.17. With the heating source located inside the sensor grid (Figure 2.14), all measurement models converge to the correct location, however both temperature measurement models exhibit rather noisy convergence. The ellipse from ultrasonic pulse one-way time of flight measurement model produces the best results with the heating source located inside the sensor grid. With the heating source located at the edge of the sensor grid (Figure 2.15), all measurement models once again converge to the correct location and both temperature measurement models and the radius from ultrasonic pulse-echo time of flight measurement model exhibit undesirable convergence behavior. The ellipse from ultrasonic pulse one-way time of flight measurement model

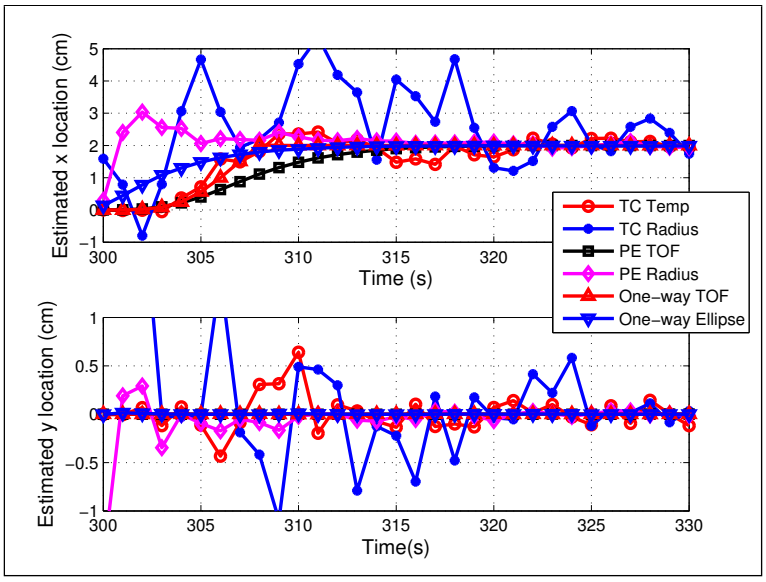

Figure 2.14 Extended Kalman filter convergence for all six measurement models with source at (2 cm, 0 cm) and initial guess of (0 cm, 0 cm).

produces the best results with the heating source located at the edge the sensor grid. With the heating source located outside of the sensor grid (Figure 2.16), none of the measurement models converge to the correct location, however the ellipse from ultrasonic pulse one-way time of flight and radius from ultrasonic pulse-echo time of flight measurement models converge to within 1 cm of the actual location. These examples started with an initial guess of (0 cm, 0 cm) for the heating source location. Figure 2.17 illustrates the convergence behavior for all six models using an initial guess of (8 cm, 8 cm) for the heating source located inside the sensor grid. Interestingly, all direct models fail to converge to the correct location in this scenario. Overall, the ellipse from ultrasonic pulse one-way time of flight measurement model produces the best results when considering accuracy of converged solution, ability to converge to the correct solution given different initial guesses, and smoothness of convergence behavior.

Because we are using numerical tools to solve the governing equations, we lack a set of state equations and cannot determine the observability index in the standard fashion. We can, however, examine sensitivity to heating source location relative to sensor location as well as sensitivity to other parameters including heating source magnitude, plate thermal conductivity, and plate surface convection coefficient. Sensitivity analysis to address observability is included in Section 2.3.

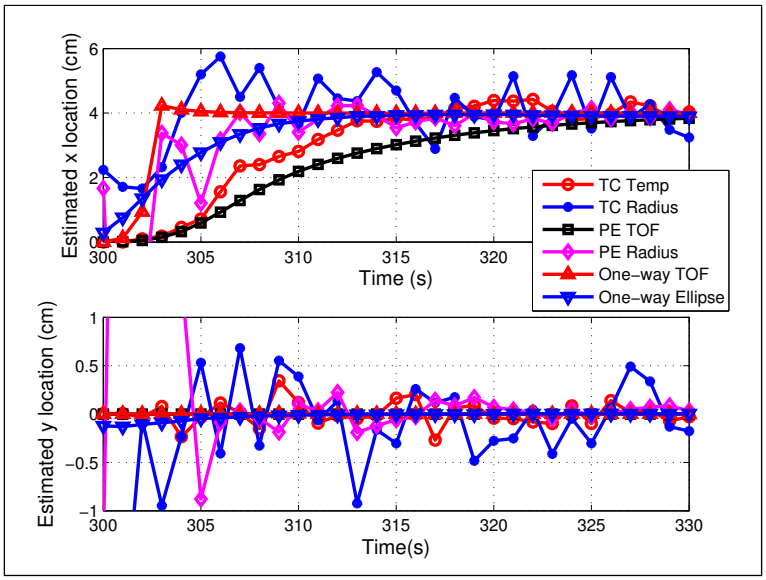

Figure 2.15 Extended Kalman filter convergence for all six measurement models with source at (4 cm, 0 cm) and initial guess of (0 cm, 0 cm).

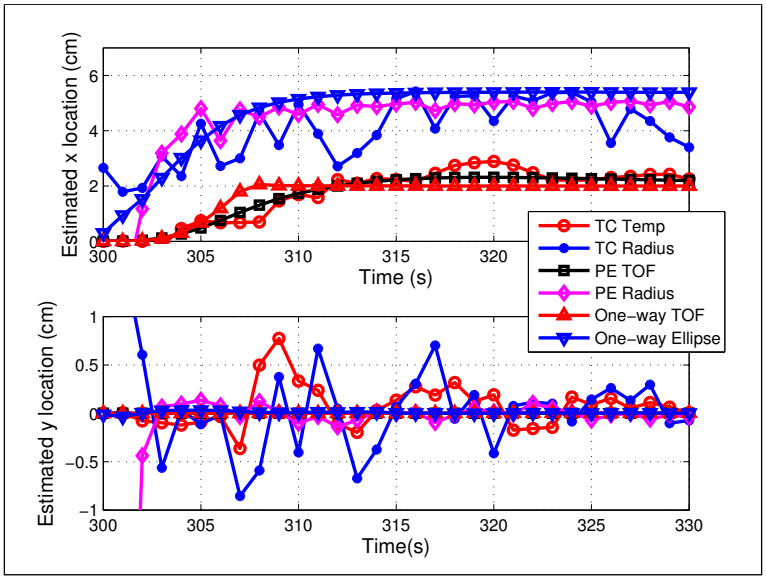

Figure 2.16 Extended Kalman filter convergence for all six measurement models with source at (6 cm, 0 cm) and initial guess of (0 cm, 0 cm).

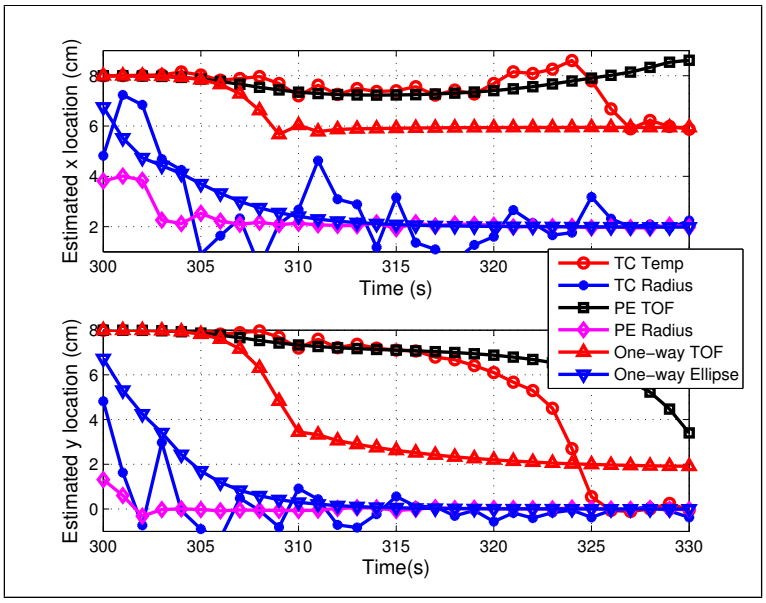

Figure 2.17 Extended Kalman filter convergence for all six measurement models with source at (2 cm, 0 cm) and initial guess of (8 cm, 8 cm).

2.2.8 Summary of convergence behavior

Results were presented from forward conduction solution development, flat plate experimentation with a known heat source, and parameter estimation of heat flux and convection coefficient on the plate. Least squares, extended Kalman filter, and extended information filter inversion methods produce similar results. This finding is significant as Section 2.4 will add more free parameters (e.g., secondary heating profile) and heat source localization to the inverse procedure. The extended Kalman filter convergence behavior is compared using six measurement models. The ellipse from ultrasonic pulse one-way time of flight measurement model produces the best results when considering accuracy of converged solution, ability to converge to the correct solution given different initial guesses, and smoothness of convergence behavior.

2.3 Sensitivity to source position, boundary conditions, and thermal conductivity

The one-way ultrasonic pulse method appears to be more robust than the other methods considered in Section 2.2. This section investigates sensitivity. Sensitivity to source position, boundary conditions, and thermal conductivity is examined in this section.

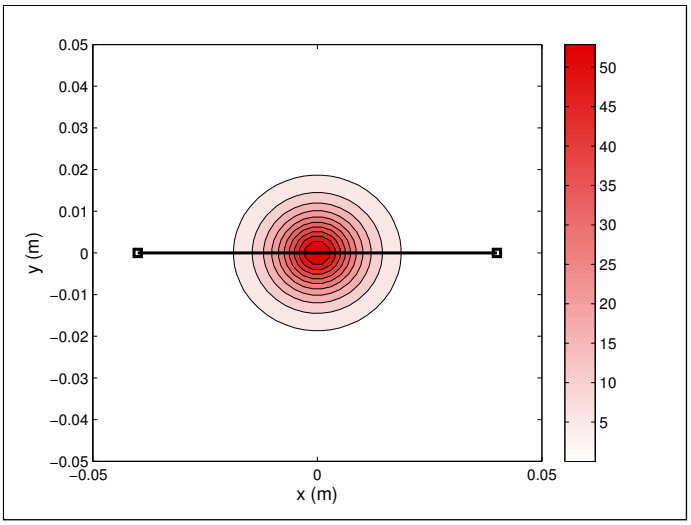

Figure 2.18 Temperature response at $t = 320\,\text{s}$ with source located at ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$). $\theta_{avg} = 13.5\,\text{K}$ between sensors.

2.3.1 Sensitivity to source position

To determine the one-way pulse method's sensitivity to source location, it is necessary to examine how the average temperature between the two sensors is affected by the relative position of the heat source. Note that this discussion assumes the source is static and therefore not moving with time. Figure 2.18 illustrates the temperature profile on the plate's non-heated side at $t = 320\,\text{s}$ with the source located at ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$). The average temperature difference between the two sensors is 13.5 K. If the source is located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$) as in Figure 2.19, the average temperature difference is nearly identical at 13.2 K. We conclude, then, that even with knowledge that the source is between the sensors, its x-location cannot be determined very accurately. If the source were further to the right, say at ($x = 3\,\text{cm}$, $y = 0\,\text{cm}$), the average temperature difference would be 11.8 K indicating that sensitivity to x-location is greater close to the transducers. However, the observations do not reveal if the heat source is closer to the pulse generator or the receiver. If the source is instead offset in the y-direction at ($x = 0\,\text{cm}$, $y = 1\,\text{cm}$) as in Figure 2.20, the average temperature difference is 5.3 K. Thus, the sensitivity to source position in the y-direction is greater than for the x-direction for this sensor pair. More generally, the sensitivity is greater in the direction normal to the ultrasonic pulse propagation path.

The sensitivity to heating source location is expressed as

$$S_{xy} = \sqrt{\left(\frac{\partial \theta_{avg}}{\partial x}\right)^2 + \left(\frac{\partial \theta_{avg}}{\partial y}\right)^2}. \tag{2.32}$$

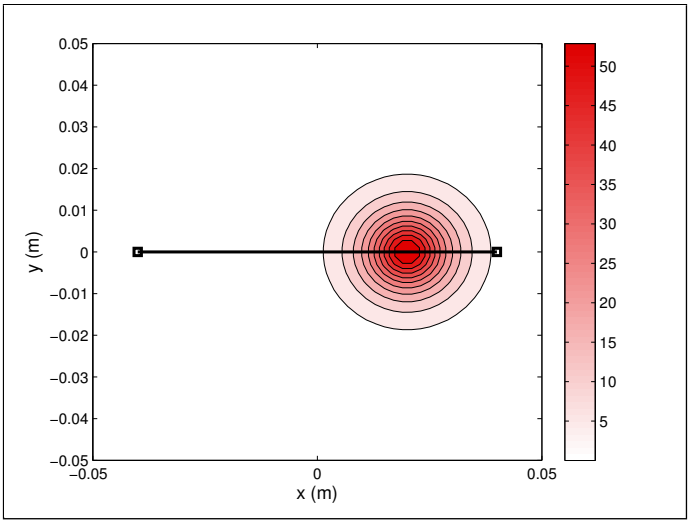

Figure 2.19 Temperature response at $t = 320\,\text{s}$ with source located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$). $\theta_{avg} = 13.2\,\text{K}$ between sensors.

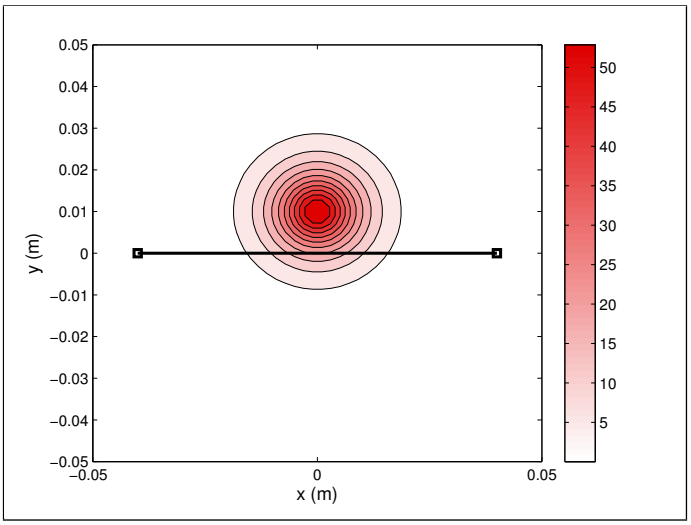

Figure 2.20 Temperature response at $t = 320\,\text{s}$ with source located at ($x = 0\,\text{cm}$, $y = 1\,\text{cm}$). $\theta_{avg} = 5.3\,\text{K}$ between sensors.

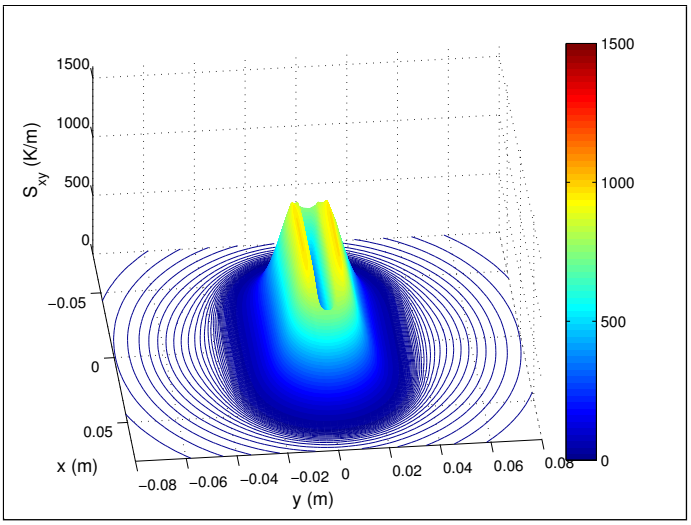

Figure 2.21 Heating source location sensitivity for one-way pulse sensor configuration and all possible heating source locations at t=320s.

Figures 2.21 and 2.22 illustrate S_{xy} at two different times during the heating. These figures highlight the high sensitivity regions around the sensor path and the drastic drop-off near the path. These figures support the observation above that sensitivity is greater perpendicular to the ultrasonic propagation path. One should notice the rapid decrease in sensitivity as the source location nears the path between sensors.

2.3.2 Sensitivity to boundary conditions and thermal conductivity

Sensitivity to boundary conditions (Figure 2.5) and thermal conductivity are analyzed for an ultrasonic sensor configuration of four sensors in an 8 cm square configuration (Figure 2.23). This configuration is based on the sensitivity analysis in Section 2.3 and represents a starting point for analysis. Sensitivity for the primary heat flux (q''), secondary heating magnitude and spread (q''_g and σ_g^2), convection coefficients for the plate (h_{sides} and h_{edges}), and thermal conductivity of the plate (k) are illustrated and analyzed for source locations inside the sensor grid and up to 2 cm outside the grid (i.e., a 12 cm by 12 cm area).

Assumptions for the sensitivity to boundary conditions and thermal conductivity analysis are:

1. Source at known, fixed position
2. Source applied at time $t = 300$ s and removed at $t = 600$ s

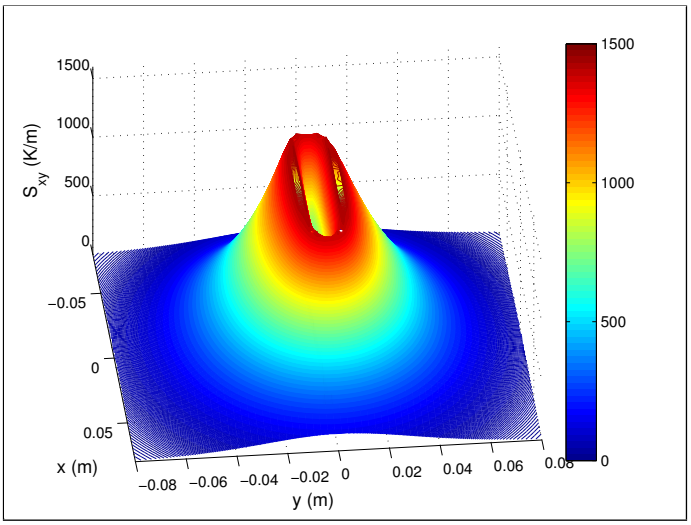

Figure 2.22 Heating source location sensitivity for one-way pulse sensor configuration and all possible source locations at $t = 450\,\mathrm{s}$.

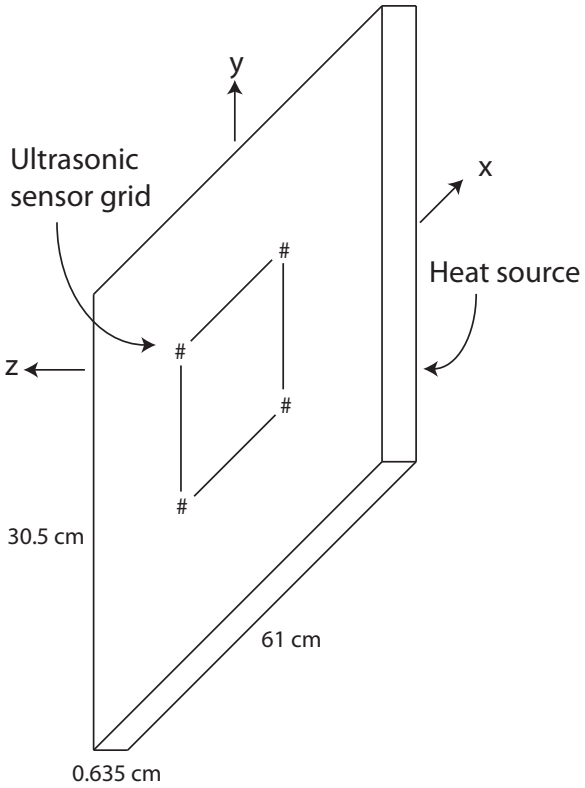

Figure 2.23 Ultrasonic sensor grid on the non-heated side of the plate with # symbols representing sensors and lines representing the ultrasonic pulse propagation paths between sensors (not drawn to scale).

3. Main heat flux $q'' = 0.930\,\mathrm{MW/m^2}$ over 0.00635 m diameter circular area during heating (value obtained in previous study [Myers et al., 2010b, Myers et al., 2010a, Myers et al., 2012b])

4. Secondary heating is characterized by a Gaussian with magnitude $q''_g = 100\,\mathrm{W/m^2}$ and variance $\sigma_g^2 = 0.0009\,\mathrm{m^2}$ during heating

5. Convection coefficient $h = 3.20\,\mathrm{W/m^2 - K}$ on both sides of the plate (value obtained in previous study [Myers et al., 2010b, Myers et al., 2010a, Myers et al., 2012b])

6. Convection coefficient $h = 3\,\mathrm{W/m^2 - K}$ on the plate edges

7. Thermal conductivity $k = 14.6\,\mathrm{W/m - K}$

8. Specific heat $C_p = 500\,\mathrm{J/kgK}$ and density $\rho = 8,000\,\mathrm{kg/m^3}$

9. Positions of sensors are (± 4 cm, ± 4 cm) on the non-heated side

Sensitivity is computed using finite differences where baseline ultrasonic pulse time of flight values are computed using the assumptions above and then new ultrasonic pulse time of flight values are computed with the parameter being investigated multiplied by $1+\delta$. Sensitivities are presented scaled according to the relation [Beck and Arnold, 1977]

$$S_{\beta_i} = \beta_i \frac{\partial G}{\partial \beta_i} \tag{2.33}$$

$$S_{\beta_i} \approx \beta_i \frac{G(x,y,z,t,\beta_1,\cdots,\beta_i(1+\delta),\cdots,\beta_p) - G(x,y,z,t,\beta_1,\cdots,\beta_p)}{\beta_i(1+\delta)-\beta_i} \tag{2.34}$$

$$S_{\beta_i} \approx \frac{G(x,y,z,t,\beta_1,\cdots,\beta_i(1+\delta),\cdots,\beta_p) - G(x,y,z,t,\beta_1,\cdots,\beta_p)}{\delta} \tag{2.35}$$

where β_i is the parameter being investigated and G is ultrasonic pulse time of flight. The δ parameter used in this section is $\delta = 0.001$. By normalizing the sensitivities, direct comparison between all investigated parameters can be performed.

2.3.2.1 Sensitivity to main heat flux (q'')

Figures 2.24 and 2.25 illustrate sensitivity to the main heat flux (q'') at $t = 320$ s and $t = 450$ s respectively. As expected from sensitivity analysis in Section 2.3, the highest sensitivity to changes in the primary heat flux are when the heating source is located near the sensor grid corners. Sensitivity is also high when the heating source is located along the ultrasonic pulse propagation path (i.e., between two sensors). Unlike the single-pair sensor approach, the sensitivity

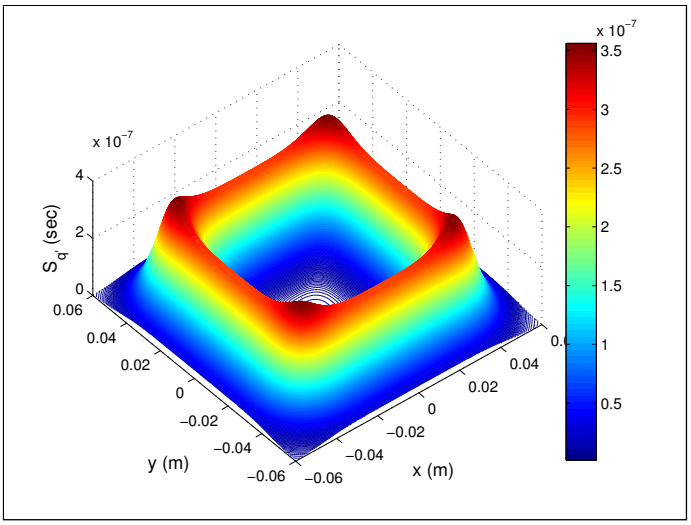

Figure 2.24 Scaled sensitivity to main heat flux ($S_{q''}$) for all possible source locations at $t = 320\,\text{s}$.

does not vanish along the propagation path because the multiple sensor paths provide additional information. Also, the difference in the sensitivity at $t = 320\,\text{s}$ and $t = 450\,\text{s}$ is due to the fact that more diffusion has occurred such that each sensor pair is able to provide more information about the average temperature rise. The peak sensitivity to the primary heat flux occurs at different times for different heating source locations. For example, if the heating source is located along the ultrasonic propagation path (Figure 2.36), the peak sensitivity to the primary heat flux occurs at the time the heating source is removed ($t = 600\,\text{s}$). If the heating source is located 4 cm above or below the propagation path (Figure 2.34), the peak sensitivity to the primary heat flux occurs approximately 60 s ($t = 660\,\text{s}$) after the heating source is removed. This latency phenomenon can be explained by the rate heat energy is conducted through the plate compared to the rate heat energy is dissipated from the plate by natural convection.

2.3.2.2 Sensitivity to secondary heat flux (q_g'' and σ_g^2)

Figures 2.26 and 2.27 illustrate sensitivity to the secondary heat flux magnitude (q_g'') at $t = 320\,\text{s}$ and $t = 450\,\text{s}$ respectively. The highest sensitivity to changes in the secondary heat flux magnitude are when the heating source is located inside the sensor grid and increases in time because thermal diffusion improves the contribution from sensor pairs far from the heat source. Figures 2.28 and 2.29 illustrate sensitivity to the secondary heat flux spread (σ_g^2) at $t = 320\,\text{s}$ and $t = 450\,\text{s}$ respectively. The highest sensitivity to changes in spread of the secondary heat flux is when the heating source is located near the center of the sensor grid. When the source is located

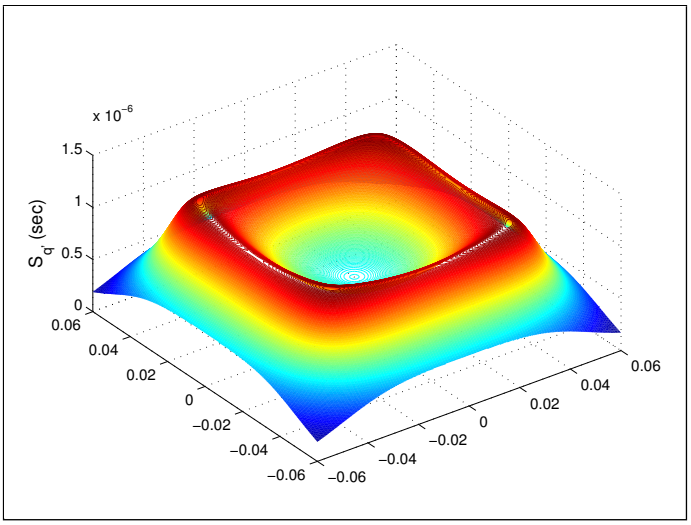

Figure 2.25 Scaled sensitivity to main heat flux ($S_{q''}$) for all possible source locations at $t = 450\,\mathrm{s}$.

along the pulse path, the spread does not affect the average temperature, so the sensitivity is small. The sensitivities to secondary heat flux magnitude and spread do not exhibit the same peak lag as with the primary heat flux. The primary heat flux is constrained to such a small area that it can be considered a point source and thus sensitivity is greatest when the point source is located along one of the ultrasonic pulse propagation paths (Figures 2.24 and 2.25). The secondary heating covers a much larger area of the plate and consequently little or no time lag is present between the energy impacting the plate and a corresponding temperature rise along all of the ultrasonic pulse propagation paths. Sensitivity to the secondary heat flux parameters should be greatest with the heating source in the center of the sensor grid where conduction effects are equal for all sensors. Figures 2.24 through 2.29 support this postulate. Sensitivity to secondary heat flux magnitude is greatest when the heating source is located inside the ultrasonic sensor grid and continues to remain high for locations outside but near the ultrasonic grid. The secondary heat flux variance parameter governs the size of the secondary heating area. The assumed profile of the secondary heating is a Gaussian with variance $\sigma_g^2 = 0.0009\,\mathrm{m}^2$. This value translates to 68% of the secondary heating energy impacting the plate over a circular area with a diameter of 6 cm with the remaining energy impacting the plate between 6 cm and 18 cm. Because of the size of the secondary heat flux, sensitivity to the spread σ_g^2 should be highest in the center of the sensor grid and the sensitivity should remain positive for all heating source locations. Figures 2.28 through 2.29 support this postulate while providing an interesting insight that sensitivity to secondary heat flux variance is lowest when the heating source is co-located with one of the ultrasonic sensors. In this situation,

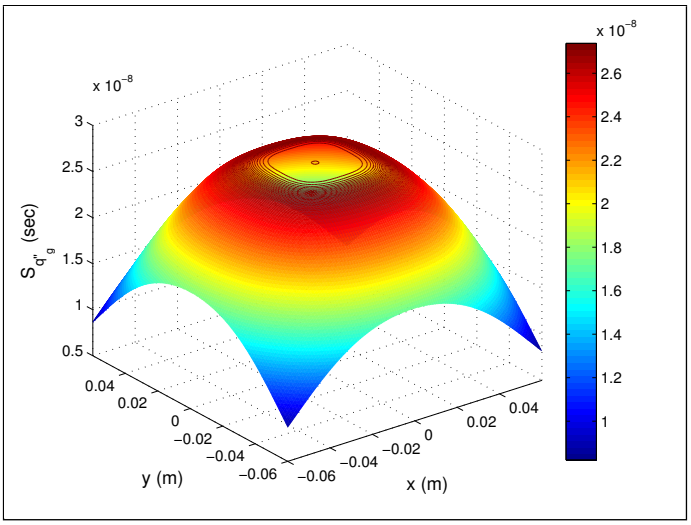

Figure 2.26 Scaled sensitivity to secondary heat flux magnitude ($S_{q''_g}$) for all possible source locations at $t = 320\,\text{s}$.

changes in the spread do not affect the plate temperature profile significantly for two of the four ultrasonic propagation paths thus causing the low sensitivity.

2.3.2.3 Sensitivity to thermal conductivity (k)

Figures 2.30 and 2.31 illustrate sensitivity to the plate's thermal conductivity (k) at $t = 320\,\text{s}$ and $t = 450\,\text{s}$ respectively. The highest sensitivity to changes in the thermal conductivity are when the heating source is located near the sensor grid corners because thermal conductivity affects plate temperatures most near the heating source. Sensitivity is also high when the heating source is located along the ultrasonic pulse propagation path (i.e., between two sensors). Thermal conductivity k is the only parameter investigated in this section that exhibits both positive and negative sensitivity. It is important to note that the sensitivity only needs to be non-zero to be usable. The sign of the sensitivity to thermal conductivity depends upon time and heating source location (Figures 2.30 and 2.31 and Figures 2.34 through 2.37). The explanation of this phenomenon is similar to the explanation for sensitivity to primary heat flux magnitude. A higher thermal conductivity results in a higher rate of conduction through the plate. For heating source locations offset from the ultrasonic pulse propagation path immediately after the heating source is applied, a higher thermal conductivity causes the energy to be conducted to the propagation path quicker resulting in higher temperatures at a given location for a given time. This temperature response results in a positive sensitivity to thermal conductivity. At some time during the experiment depending on the heating

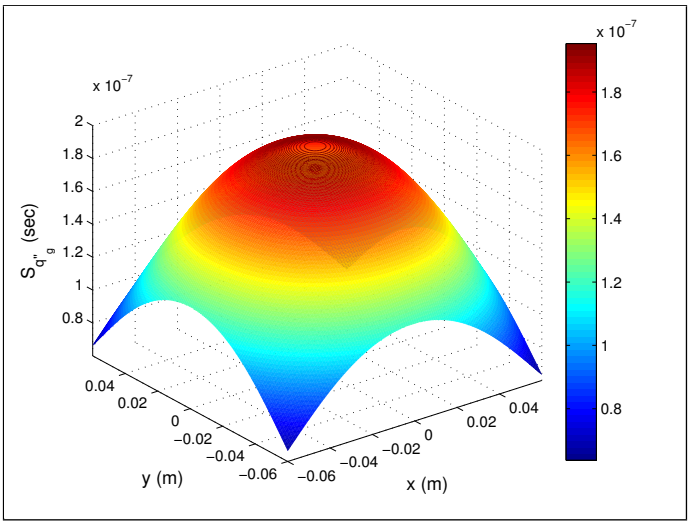

Figure 2.27 Scaled sensitivity to secondary heat flux magnitude ($S_{q''_g}$) for all possible source locations at $t = 450\,\mathrm{s}$.

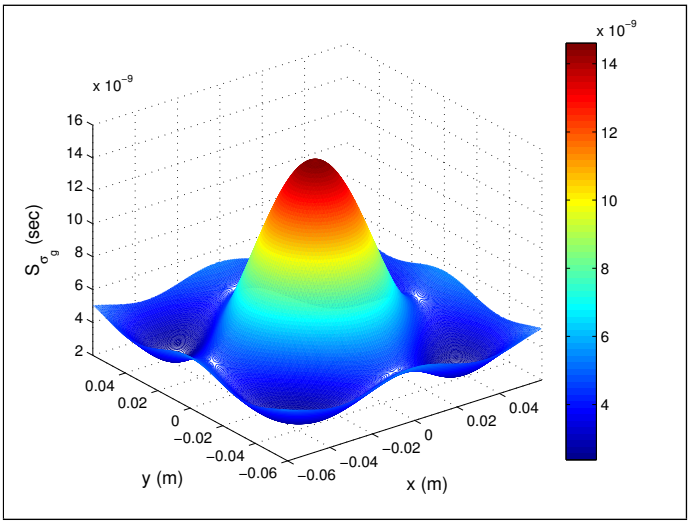

Figure 2.28 Scaled sensitivity to secondary heat flux variance ($S_{\sigma_g^2}$) for all possible source locations at $t = 320\,\mathrm{s}$.

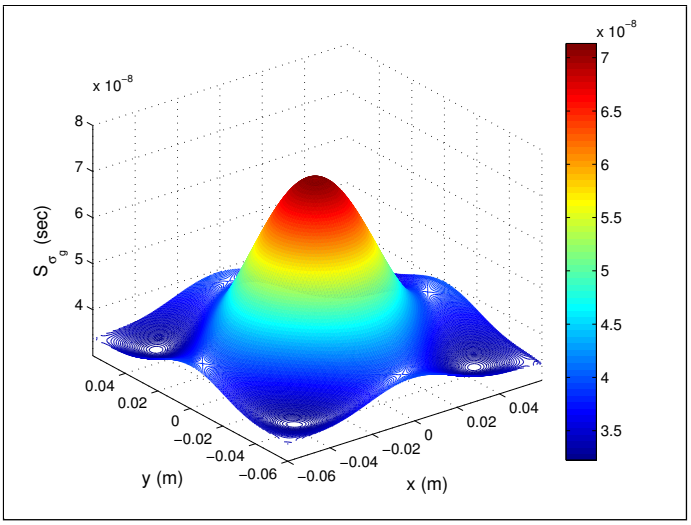

Figure 2.29 Scaled sensitivity to secondary heat flux variance ($S_{\sigma_g^2}$) for all possible source locations at $t = 450\,\text{s}$.

source location, sensitivity to thermal conductivity turns negative because the temperature distribution in the plate is more uniform. In other words, more heat energy is conducted away from the heating source location. Figure 2.36 illustrates this point because the heating source is located along the ultrasonic pulse propagation path. More heat energy is conducted away from the propagation path starting when the heating source is applied resulting in a lower average temperature along the path, which implies lower sensitivity.

2.3.2.4 Sensitivity to convection coefficient on the plate sides (h_{sides})

Figures 2.32 and 2.33 illustrate sensitivity to the convection coefficient on the plate sides (h_{sides}, i.e., the heated and non-heated side of the plate) at $t = 320\,\text{s}$ and $t = 450\,\text{s}$ respectively. The highest sensitivity to changes in h_{sides} are when the heating source is located near the sensor grid corners. Sensitivity is also high when the heating source is located along the ultrasonic pulse propagation path. We can understand this effect as a response to a global increase or decrease in plate heated-zone temperature. As the heat transfer coefficient increases, plate temperatures decrease and the amount of cooling will be more pronounced at the locations of highest temperature. Therefore, the sensitivities will be largest when the highest temperatures are located at the sensor grid corners and along the pulse paths. Comparing the illustrations for $t = 320\,\text{s}$ and $t = 450\,\text{s}$, sensitivity to h_{sides} becomes greater at longer times and sensitivity is highest when the heating source is located inside the sensor grid. Sensitivity to the convection coefficient on the plate sides h_{sides} is

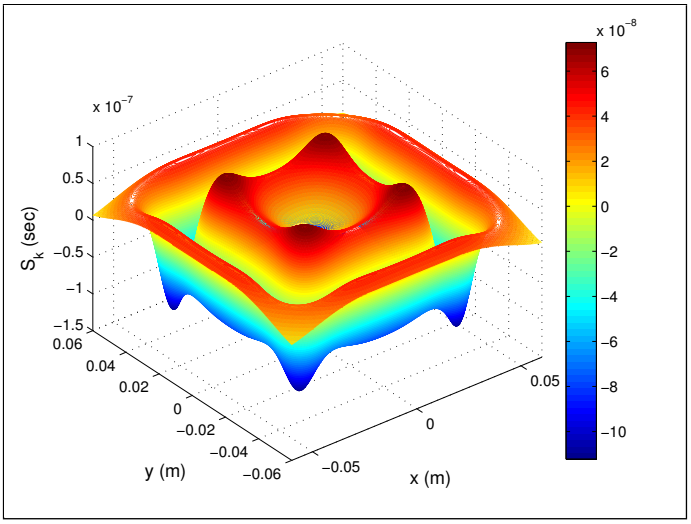

Figure 2.30 Scaled sensitivity to thermal conductivity (S_k) for all possible source locations at $t = 320\,\mathrm{s}$.

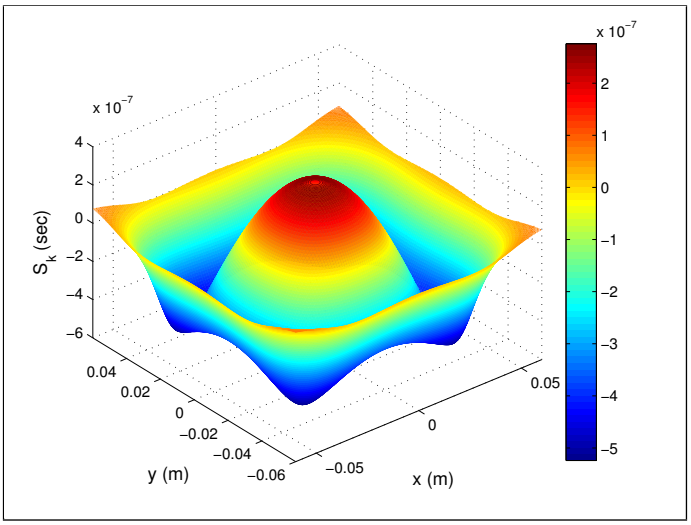

Figure 2.31 Scaled sensitivity to thermal conductivity (S_k) for all possible source locations at $t = 450\,\mathrm{s}$.

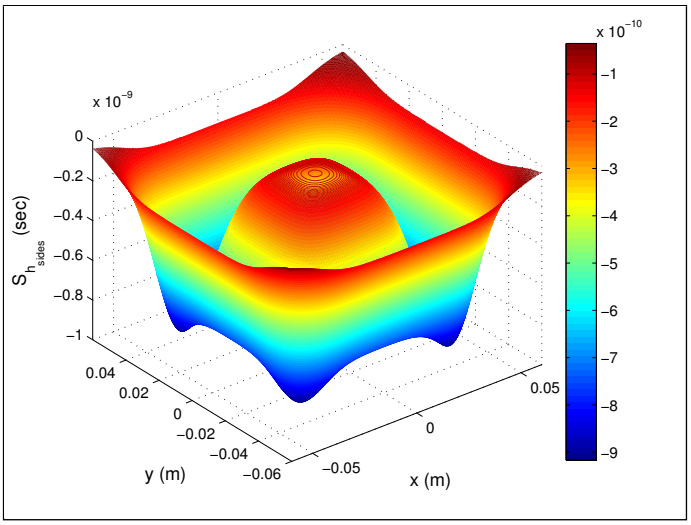

Figure 2.32 Scaled sensitivity to the convection coefficient on the plate sides ($S_{h_{sides}}$) for all possible source locations at $t = 320\,\text{s}$.

negative throughout the experiment since an increased convection reduces plate temperatures.

2.3.2.5 Sensitivity to convection coefficient on the plate edges (h_{edges})

Sensitivity to the convection coefficient on the plate edges ($S_{h_{edges}}$) is smaller than the numerical precision of the solution. Illustrations, therefore, are not presented here but are included in the combined sensitivity plots in the next section.

2.3.2.6 Summary of sensitivity to boundary conditions and thermal conductivity

Figure 2.34 illustrates the sensitivities for each parameter for all times during the experiment with the heating source in the center of the sensor grid. While sensitivity to primary heat flux is the largest, sensitivities to secondary heating magnitude and thermal conductivity are on the same order of magnitude, and sensitivities to secondary heating spread and convection on the plate sides are one order of magnitude smaller. Sensitivities to primary heat flux, secondary heating magnitude, and secondary heating spread are correlated and the sensitivities to secondary heating spread and convection on the plate sides and edges are much smaller ($< 10\%$) than the sensitivity to the primary heat flux. This means any inverse routine will be able to estimate q and k reliably, but the others will have large confidence intervals and simultaneous estimation of these parameters will be difficult. It should be noted, however, that the sensitivities to the primary heat flux and thermal conductivity are correlated for approximately the first 70 s of heating making simultaneous

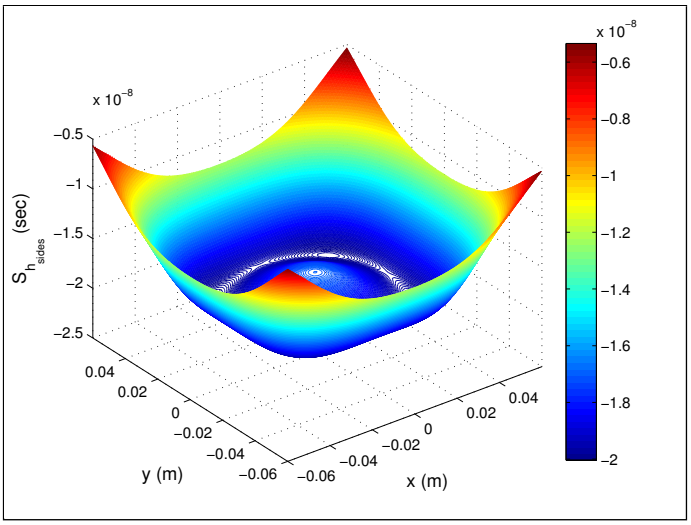

Figure 2.33 Scaled sensitivity to the convection coefficient on the plate sides ($S_{h_{sides}}$) for all possible source locations at $t = 450\,\text{s}$.

estimation of q and k during this time difficult.

Figure 2.35 illustrates the sensitivities for each parameter for all times during the experiment with the heating source offset from the center of the sensor grid by 2 cm in the y-direction. Sensitivity to thermal conductivity during heating is smaller than with the heating source in the center of the sensor grid. During the cool-down phase, sensitivity to thermal conductivity appears quite similar to the sensitivity when the heating source is in the center of the sensor grid. The remaining parameters exhibit similar sensitivities when the heating source is located in the center of the sensor grid and when the heating source is offset by 2 cm. Again, for this heating location we see large, uncorrelated sensitivities for q and k and the correlated portion at heating start is reduced from approximately 70 s to approximately 25 s. Thus, any inverse routine will be able to estimate q and k reliably after 25 s of heating, but the others parameters will have large confidence intervals and simultaneous estimation of these parameters will be difficult.

Figure 2.36 illustrates the sensitivities for each parameter for all times during the experiment with the heating source offset from the center of the sensor grid by 4 cm in the y-direction which puts the heating source directly between one sensor pair. As expected, sensitivity to the primary heat flux is slightly higher and decreases immediately after heating stops. Sensitivity to thermal conductivity remains negative throughout the experiment and does not cross the axis as it does when the heating source is not directly between a sensor pair. Sensitivity to the secondary heating variance is slightly lower. In this case with the heating source directly between one sensor pair, only one parameter can be reliably estimated at one time since all of the sensitivities are

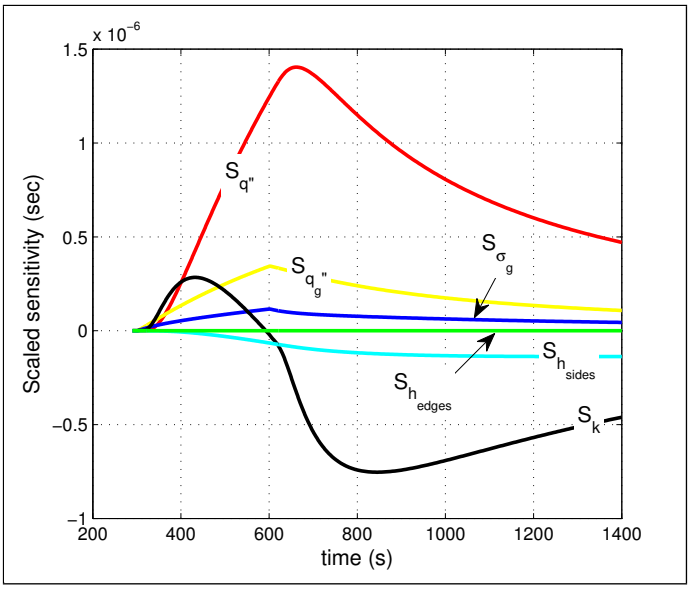

Figure 2.34 Scaled sensitivity for six parameters with heating source located at ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

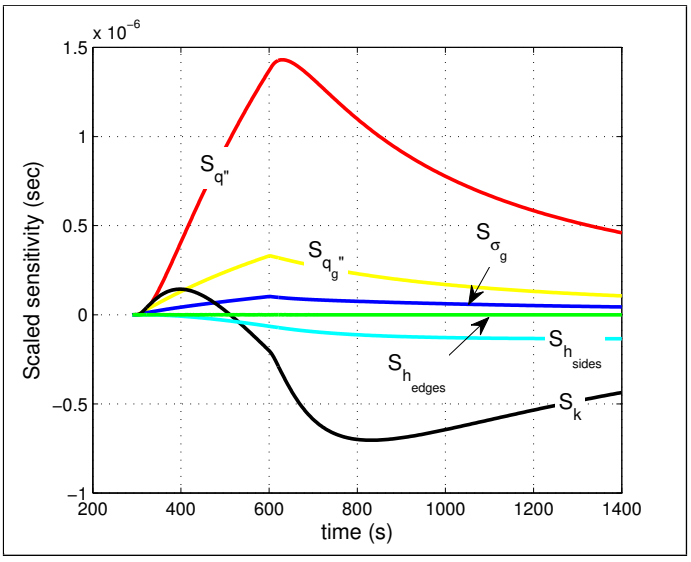

Figure 2.35 Scaled sensitivity for six parameters with the heating source located at (0 cm, 2 cm).

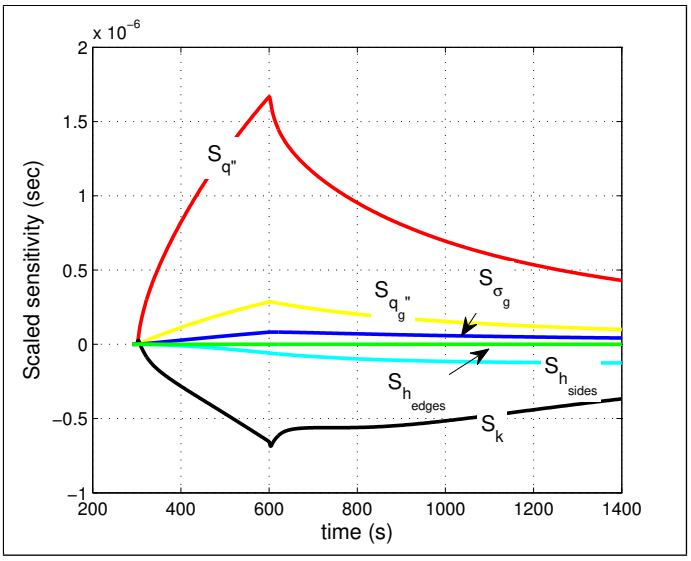

Figure 2.36 Scaled sensitivity for six parameters with the heating source located at ($x = 0$ cm, $y = 4$ cm).

correlated or are too small ($< 10\%$ of primary heat flux sensitivity).

Figure 2.37 illustrates the sensitivities for all parameters and for all times during the experiment with the heating source offset from the center of the sensor grid by 6 cm in the y-direction which puts the heating source outside of the sensor grid. The sensitivities are similar to the 2 cm offset, however the magnitudes range from 30% to 50% lower. For this heating location we see large, uncorrelated sensitivities for q and k and the correlated portion at heating start is approximately 25 s. Thus, any inverse routine will be able to estimate q and k reliably with the heating source in this location, but the other parameters will have large confidence intervals and simultaneous estimation of these parameters will be difficult.

2.4 Parameter estimation to locate static spot heating source

Justification for using the extended Kalman filter is presented in Section 2.1 when estimating the unknown boundary conditions present during the experiment detailed in Section 2.1.1. To further justify use of the extended Kalman filter, this section presents analysis on estimating the location of the spot heating source using the extended Kalman filter, the particle filter, and ordinary least squares methods. An experiment using ultrasonic transducers instead of thermocouples is used to provide ultrasonic pulse time of flight data to the parameter estimation.

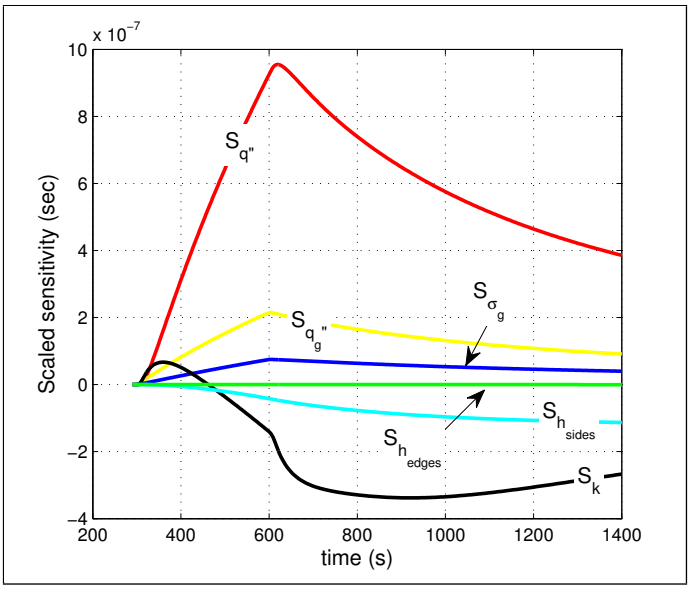

Figure 2.37 Scaled sensitivity for six parameters with the heating source located at ($x = 0\,\text{cm}$, $y = 6\,\text{cm}$).

2.4.1 Flat plate experiment using ultrasonic transducers

Consider a $61\,\text{cm} \times 30.5\,\text{cm} \times 0.635\,\text{cm}$ stainless steel 316L plate (Figure 2.1) with constant properties (Table 2.1). The plate is large enough so the plate edges do not affect the temperature profile in the plate during the experiment. The heating source, a Research, Inc. SpotIR® 4150 heater with focusing cone, is positioned approximately 2 mm from the plate surface such that its beam strikes a fixed position on the plate and is applied at $t = 300\,\text{s}$ and removed at $t = 600\,\text{s}$. A parameter estimation study concluded the SpotIR® heater has a heating profile of $q'' = 0.930\,\text{MW/m}^2$ over 0.635 cm diameter circular area with a secondary heating modeled as a Gaussian with a profile of $q''_g = 100\,\text{W/m}^2$ and a variance of $\sigma_g^2 = 0.0009\,\text{m}^2$ (Figure 2.5) [Myers et al., 2010b, Myers et al., 2012b, Myers et al., 2012c]. The study also concluded the convection coefficient on the plate sides is $h = 3.20\,\text{W/m}^2 - \text{K}$. The convection coefficient on the plate edges is assumed to be $h = 3\,\text{W/m}^2 - \text{K}$ and radiation losses from the plate are assumed to be negligible.

Two ultrasonic sensors consisting of Ferroperm Piezoceramics Pz46 2 MHz transducer elements measuring 10 mm in diameter and 1 mm thick are bonded to the non-heated side of the plate using silver epoxy. With plate center on the heated side being the origin and the x-axis being the length (Figure 2.1), transducers are attached at ($x = -4\,\text{cm}$, $y = 0\,\text{cm}$) and ($x = 4\,\text{cm}$, $y = 0\,\text{cm}$) locations on the non-heated side ($z = 0.635\,\text{cm}$). One transducer transmits ultrasonic pulses while the other transducer receives the pulses and time of flight is recorded.

Separate experiments are conducted with the source positioned on the heated side of the plate at (x,y) locations of (0 cm, 0 cm), (0 cm, 2 cm), (0 cm, 4 cm), (0 cm, 6 cm), (0 cm, 8 cm), and (0 cm, 10 cm). Black Zynolyte® Hi-Temp Paint is applied to a 1.5 cm wide strip at the plate center to maximize energy absorption from the heater. The plate is oriented vertically with the positive y-axis pointing up. Data acquisition equipment employing cross-correlation techniques is used to determine and record ultrasonic pulse time of flight readings once per second during the experiment.

These experiments were conducted by Industrial Measurement Systems, Inc. in Aurora, IL.

2.4.2 3D conduction solution

The forward conduction solution used in this localization study is developed in Section 2.4.2 and leverages COMSOL Multiphysics® by the COMSOL Group and MATLAB® by The Mathworks, Inc. Figures 2.38 and 2.39 illustrate the agreement between the COMSOL® model and the ultrasonic time of flight measured during the experiment. The residuals [Beck and Woodbury, 1998, Dowding and Blackwell, 2001] provide valuable insight into the accuracy of the model and indicate that the solution is somewhat biased. Agreement between the model and the experiment is acceptable; however, the magnitude with the heat source located at ($x = 0$ cm, $y = 0$ cm) and when the heat source is located at ($x = 0$ cm, $y = 2$ cm) are both underestimated by the model. Figure 2.40 illustrates the time of flight measurements during the beginning part of the experiment and highlights the time needed for the heat to reach the sensors.

2.4.3 Inverse methods comparison

This section examines heating source localization using four ultrasonic transducers in an 8 cm square pattern (Figure 2.41). Multiplexing equipment was not available when this study was conducted; therefore, data from separate experiments detailed in Section 2.4.1 are used together in this section to simulate the four sensor array.

Locating and characterizing the boundary layer transition depends upon many factors such as heating source movements in time, heating source magnitude changes in time, and other transient behaviors. Fairly restrictive assumptions can be imposed that simplify the problem. Analysis and algorithm development can proceed using these restrictive assumptions and then assumptions can be relaxed in stages to achieve the end result of source localization and characterization. The assumptions for this work are:

1. Source in fixed position (location unknown)

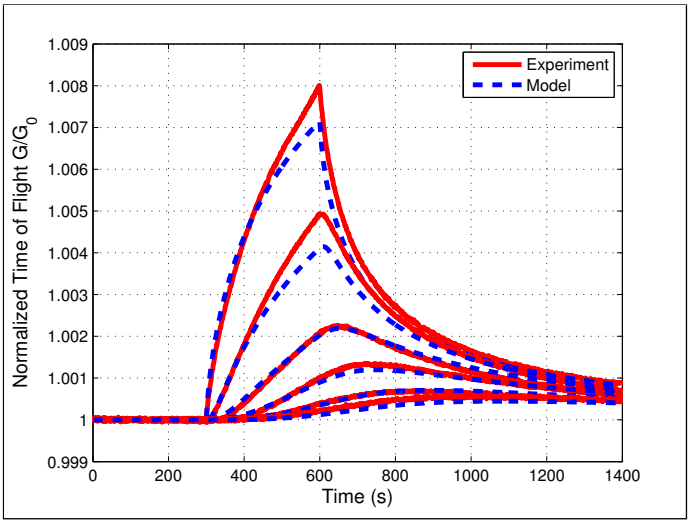

Figure 2.38 Comparison of the COMSOL® model with the one-way ultrasonic pulse experiment with heat source located between the sensors (top curve) and offset by 2 cm, 4 cm, 6 cm, 8 cmm, and 10 cm. The model uses temperatures along the non-heated surface of the plate.

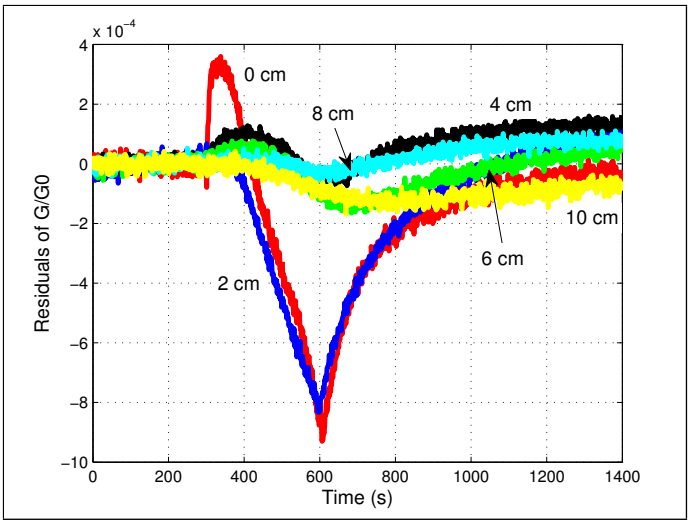

Figure 2.39 Residuals between the COMSOL® model and the one-way ultrasonic pulse experiment. The model uses temperatures along the non-heated surface of the plate.

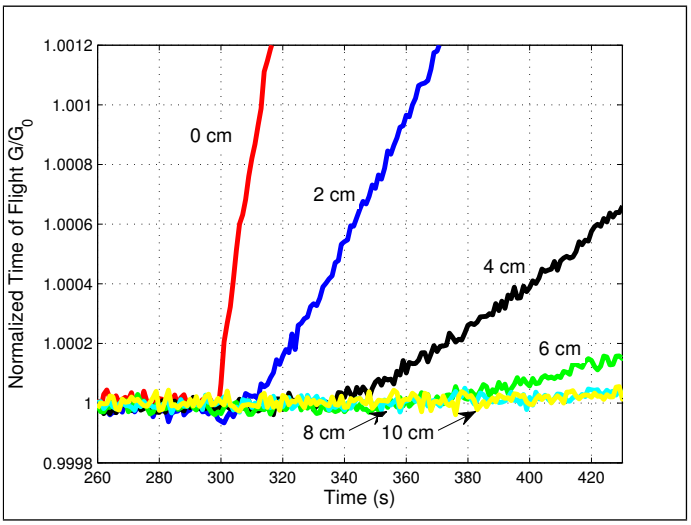

Figure 2.40 One-way ultrasonic pulse time of flight measurements for the beginning part of the heating phase.

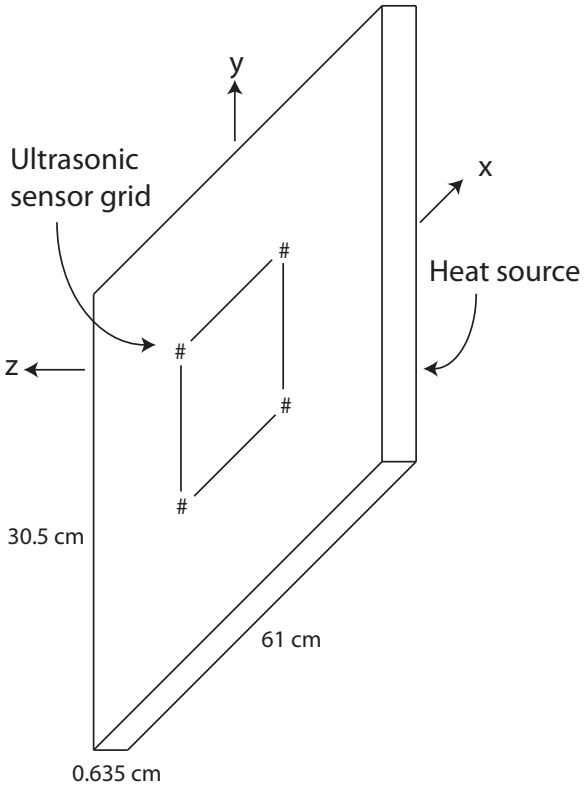

Figure 2.41 Ultrasonic sensor grid on the non-heated side of the plate with # symbols representing sensors and lines representing the ultrasonic pulse propagation paths between sensors (not drawn to scale).

2. Source applied at time $t = 300\,\mathrm{s}$ and removed at $t = 600\,\mathrm{s}$

3. Main heat flux $q'' = 0.930\,\mathrm{MW/m^2}$ over $0.00635\,\mathrm{m}$ diameter circular area while source applied (value obtained in previous study [Myers et al., 2010b, Myers et al., 2010a, Myers et al., 2012b])

4. Secondary heating is characterized by a Gaussian with magnitude $q''_g = 100\,\mathrm{W/m^2}$ and spread $\sigma_g^2 = 0.0009\,\mathrm{m^2}$ while source applied

5. Convection coefficient $h = 3.20\,\mathrm{W/m^2K}$ on both sides of the plate (value obtained in previous study [Myers et al., 2010b, Myers et al., 2010a, Myers et al., 2012b])

6. Convection coefficient $h = 3\,\mathrm{W/m^2K}$ on the plate edges

7. Thermal conductivity $k = 14.6\,\mathrm{W/mK}$

8. Specific heat $C_p = 500\,\mathrm{J/kgK}$ and density $\rho = 8{,}000\,\mathrm{kg/m^3}$

9. Positions of sensors are ($\pm 4\,\mathrm{cm}$, $\pm 4\,\mathrm{cm}$) on the non-heated side

The three inverse methods compared in this work are: extended Kalman filter, particle filter, and least squares. The two measurement models studied are: ultrasonic pulse one-way time of flight measurement model, and ellipse from ultrasonic pulse one-way time of flight measurement model. With the particle filter, only the ultrasonic pulse one-way time of flight measurement model is considered because, as will become clear later in this work, the particle filter does not depend upon a Jacobian, and the ellipse model is developed specifically as an alternative way of expressing the Jacobian for those methods that require a Jacobian. Comparison of the five methods is performed in locating the heating source on the plate in the $x-y$ plane (x_q, y_q). For all methods, the state therefore is $X_t = [x_q, y_q]^T$. In all methods considered in this work, the ultrasonic time of flight is normalized by the time of flight before the heating source is applied to the plate (G_{ij}/G_0).

2.4.3.1 Extended Kalman filter with ultrasonic pulse one-way time of flight measurement model

The extended Kalman filter algorithm to locate the source can be found in Table 2.2 . The state is $X_t = [x_q, y_q]^T$, and there is no input (U_t) to the state; thus the extended Kalman filter state model is $a = I_2$ and the state model Jacobian is $A = I_2$, where I_2 is a 2×2 identity matrix. A parameter sweep is conducted for the state model variance from $\sigma^2 = 0.01\,\mathrm{m^2}$ to $\sigma^2 = 0.000001\,\mathrm{m^2}$. A small value for the state model variance ($\sigma^2 = 0.000001\,\mathrm{m^2}$) produces a damping effect on the convergence whereas a large value ($\sigma^2 = 0.01\,\mathrm{m^2}$) produces fast but erratic convergence. From

this parameter sweep, it is determined that a state variance of $\sigma^2 = 0.0001\,\mathrm{m}^2$ provides a good compromise between damping and stability and this value is used in this work. Thus, the state model covariance matrix is $Q_t = 0.0001\,\mathrm{m}^2 \times I_2$.

This measurement model consists of obtaining expected temperatures from COMSOL® using the predicted state $\overline{X}_t$, computing the average temperature between the transducers, and then computing an expected time of flight from 3.4 to form $b(\overline{X}_t)$ (equation 2.36). For the current analysis, the average temperature is computed along the path on the non-heated plate surface between the two sensors. The Jacobian partial derivatives are obtained using finite difference when moving the source in the x and y directions independently (equation 2.37).

$$b(\overline{X}_t) = \begin{bmatrix} \overline{G}_1 \\ \overline{G}_2 \\ \overline{G}_3 \\ \overline{G}_4 \end{bmatrix}; \tag{2.36}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{G}_1}{\partial x_1} & -\frac{\partial \overline{G}_1}{\partial y_1} \\ -\frac{\partial \overline{G}_2}{\partial x_2} & -\frac{\partial \overline{G}_2}{\partial y_2} \\ -\frac{\partial \overline{G}_3}{\partial x_3} & -\frac{\partial \overline{G}_3}{\partial y_3} \\ -\frac{\partial \overline{G}_4}{\partial x_4} & -\frac{\partial \overline{G}_4}{\partial y_4} \end{bmatrix}, \tag{2.37}$$

where t is time in seconds with a time step of 1 s, $\overline{G}_i$ with $i = 1,2,3,4$ is the ultrasonic pulse time of flight with the heating source located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ are the locations of four transducers. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience because this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the x and y positions of all sensors by 0.0001 m. Based on the flat plate experiment above, the sensor noise is assumed be $\pm 6 \times 10^{-5}$ (a non-dimensional number based on G_{ij}/G_0) and is normally distributed ($\sigma^2 = ((6 \times 10^{-5})/3)^2 = 4 \times 10^{-10}$). Solution instabilities were present when using this variance, which were reduced by increasing the variance to 4×10^{-7}. This larger variance effectively dampens the solution and prevents large changes from one iteration to the next. The measurement covariance matrix, therefore, is $R = 4 \times 10^{-7} \times I_4$.

2.4.3.2 Extended Kalman filter with ellipse from ultrasonic pulse one-way time of flight measurement model

In an attempt to simplify the sensitivity calculation, the lines of constant time of flight around the sensor pairs form approximate ellipses. With this approximation, the sensitivities can be

calculated algebraically. Figure 2.13 illustrates the geometry of an ellipse, where the two sensors are assumed to be the foci for the ellipse. Since the distance between sensors is known, ellipse parameters c and d can be related to each other and the ellipse can be represented with just one parameter c.

$$r_{is} + r_{js} = 2c = \sqrt{r_{ij}^2 + 4d^2} \tag{2.38}$$

where i and j are sensors and s is heat source.

$$c = \frac{1}{2}\sqrt{r_{ij}^2 + 4d^2} = \frac{r_{is} + r_{js}}{2} \tag{2.39}$$

$$r_{is} = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{2.40}$$

$$r_{js} = \sqrt{(x_j - x_s)^2 + (y_j - y_s)^2} \tag{2.41}$$

$$\frac{\partial c_i}{\partial x_s} = \frac{1}{2}\left[\frac{x_s - x_i}{r_{is}} + \frac{x_s - x_j}{r_{js}}\right] \tag{2.42}$$

$$\frac{\partial c_i}{\partial y_s} = \frac{1}{2}\left[\frac{y_s - y_i}{r_{is}} + \frac{y_s - y_j}{r_{js}}\right] \tag{2.43}$$

This measurement model consists of measuring the one-way ultrasonic pulse time of flight, using COMSOL® to obtain the average temperature between the transducers for a range of c values, using equation 3.4 to compute time of flight for the range of c values, and then interpolating the time of flight results using the spline method to obtain c for the measured time of flight. The measurement transition function $b(\overline{X}_t)$ and the Jacobian B_t are then

$$b(\overline{X}_t) = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}; \tag{2.44}$$

$$B_t = \begin{bmatrix} \frac{\partial c_1}{\partial x_s} & \frac{\partial c_1}{\partial y_s} \\ \frac{\partial c_2}{\partial x_s} & \frac{\partial c_2}{\partial y_s} \\ \frac{\partial c_3}{\partial x_s} & \frac{\partial c_3}{\partial y_s} \\ \frac{\partial c_4}{\partial x_s} & \frac{\partial c_4}{\partial y_s} \end{bmatrix}, \tag{2.45}$$

where t is time in seconds with a time step of 1 second, c_i with $i = 1,2,3,4$ is the ellipse parameter if the source is located at (x_s, y_s). Based on the flat plate experiment above, sensor noise is assumed be $\pm 1.05 \times 10^{-8}$ s and is normally distributed ($\sigma^2 = 1.22 \times 10^{-17}$ sec^2). The sensor noise in terms

Table 2.4 Particle filter algorithm.

Step	Operation
1	Generate m random possible source locations across the plate (particles)
2	Obtain expected measurements for each particle i for the current time t ($Z_{i,t}$)
3	Obtain actual measurements for the current time ($Ztrue_t$)
4	Weight each particle i according to $N(Ztrue_t, I)$
5	Normalize weights for each particle i into bins from 0 to 1
6	Resample best particles using normal distribution
7	Add position noise to each particle
8	Return to Step 2 for next time step

of temperature can be expressed as

$$\theta_{noise} = \frac{G_{noise} v_0}{L\xi} = 6.09\,\mathrm{K} \tag{2.46}$$

Using the average slope of $0.015\,\mathrm{m/K}$ determined in Section 2.2 and documented in previous work [Myers et al., 2010b,Myers et al., 2010a,Myers et al., 2012b], ellipse noise for the c parameter from ultrasonic pulse time of flight measurement model is $\pm 0.0914\,\mathrm{m}$ and is normally distributed ($\sigma^2 = 9.28 \times 10^{-4}\,\mathrm{m}^2$). Since the measurement covariance matrix R represents the measurement noise of the c parameter, the measurement covariance is $3.19 \times 10^{-10}\,\mathrm{m}^2$ ($[1.05 \times 10^{-8}\,\mathrm{s}/3 \times 5,100\,\mathrm{m/sec}$ sound speed$]^2$).

2.4.3.3 Particle filter with ultrasonic pulse one-way time of flight measurement model

The particle filter is an alternative nonparametric implementation of the Bayes filter and is a Monte Carlo technique used for the solution of state estimation problems. The main idea is to represent the required posterior density function by a set of random samples with associated weights and to compute the estimates based on these samples and weights [Vianna et al., 2010]. Because it is nonparametric, the particle filter can represent a much broader space of distributions than Gaussians and has the ability to model nonlinear transformations of random variables [Thrun et al., 2006]. The particle filter algorithm to locate the source can be found in Table 2.4.

Implementation of the particle filter for heating source localization starts with defining the area of possible source locations on the plate. The number of particles m to use in the algorithm must also be defined. A large number of particles yields a higher probability that one or more particles will be located near the actual source but the downside is higher computational cost. For this study, the area is defined as the $8\,\mathrm{cm} \times 8\,\mathrm{cm}$ sensor grid and the number of particles is $m = 40$.

The algorithm starts with the generation of m random particles within the defined area of possible source locations (Figure 2.42). Step 2 involves obtaining expected temperatures from COMSOL® with the heating source at each particle location, computing the average temperature between the transducers, and then computing an expected time of flight to form $Z_{i,t}$, a 4×1 matrix for each particle i at the current time t. For the current analysis, the average temperature is computed along the path on the non-heated plate surface between the two sensors. Step 3 entails obtaining actual ultrasonic time of flight measurements at the current time t for all four sensor pairs to form $Ztrue_t$, a 4×1 matrix. The particle filter relies on an importance factor, or weight $w_{i,t}$, to incorporate the measurement $Ztrue_t$ into the particle set. The weight $w_{i,t}$ for each particle is computed in Step 4 using

$$\begin{aligned} w_{i,t} &= N(Ztrue_t, I) \\ &= det(2\pi I)^{-\frac{1}{2}} exp\left\{ -\frac{1}{2}((Z_{i,t} - Ztrue_t) \times gain)^T I^{-1}((Z_{i,t} - Ztrue_t) \times gain) \right\}. \end{aligned} \tag{2.47}$$

Gain is discussed in detail later in this section. A number of resampling techniques have been devised [Thrun et al., 2006, Vianna et al., 2009, Arulampalam et al., 2002]. This work employs the sampling importance resampling technique because this technique requires fewer particles than some of the other methods and focuses the computational resources to regions in the state space with high posterior probability [Thrun et al., 2006]. In Step 5, the particle weights are normalized into bins from 0 to 1, which gives the particles with the highest weight the largest bins and then in Step 6, the particles are resampled using a normal distribution. By resampling across the bins from 0 to 1 using a normal distribution, a higher probability exists that the best particles will be chosen but some particles that are not the best will be chosen too. It is important to note that the number of particles m remains constant through the resampling process, thus some particles will have identical locations on the plate after resampling. Degeneracy is common with particle filters, a situation where the solution converges to the one best particle within the current particle set without considering locations nearby [Doucet et al., 2002]. This fact necessitates Step 7 where position noise or roughness is added to each particle [Salmond et al., 1993]. In this work, uniform position noise of ± 0.5 cm is used. Figures 2.43 and 2.44 illustrate the particle distribution before and after adding noise to each particle location. After completion of Step 7, the algorithm returns to Step 2 and the process is repeated for the next step in time. For this work, a time step of 1 s is used. Figure 2.45 illustrates the particle distribution at $t = 330$ sec.

Because the magnitude of the non-dimensional values in the matrix $Z_{i,t} - Ztrue_t$ in equation 2.47 ranges from 0 to 0.008, using no *gain* ($gain = 1$) produces a value of 1 in the exponent portion of the equation for all particles. Thus, identical weights are computed for all particles (Figure 2.46). For the particle filter to function properly, it is imperative that the particles close to

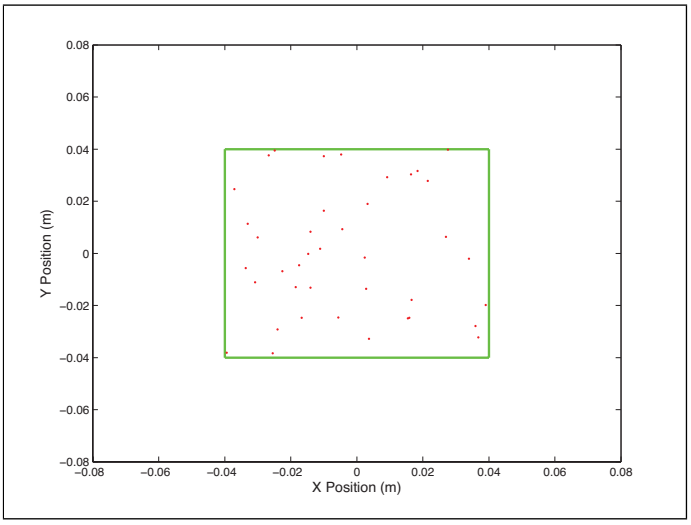

Figure 2.42 Particle locations ($m = 40$) at 300 s.

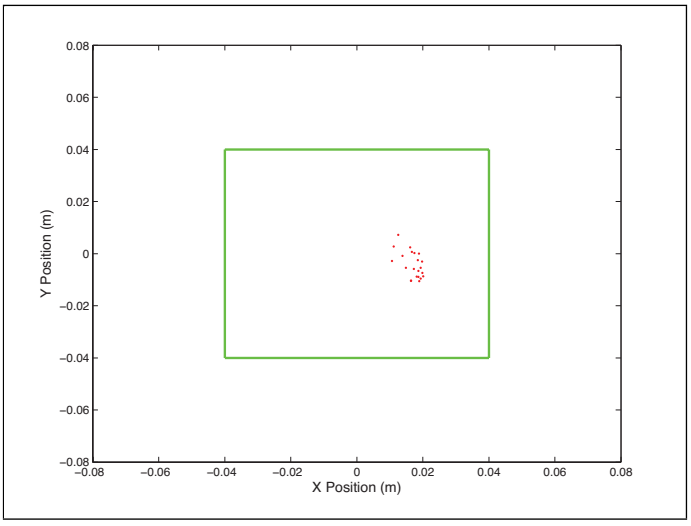

Figure 2.43 Particle locations ($m = 40$) at 315 s.

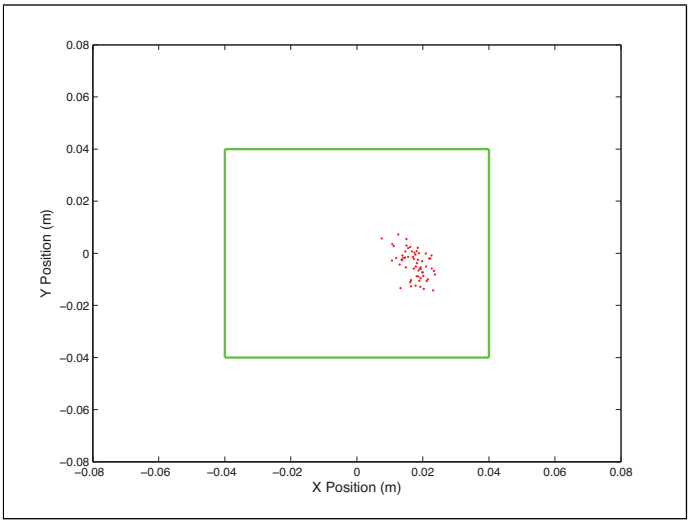

Figure 2.44 Particle locations ($m = 40$) at 315 s after adding noise to each particle location.

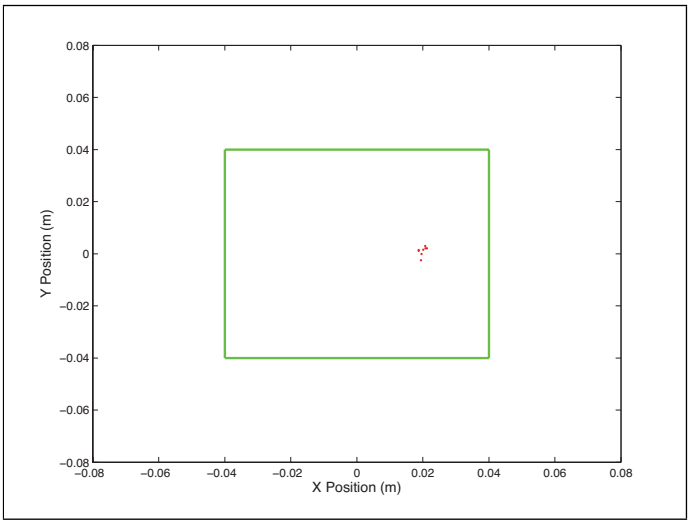

Figure 2.45 Particle locations ($m = 40$) at 330 s.

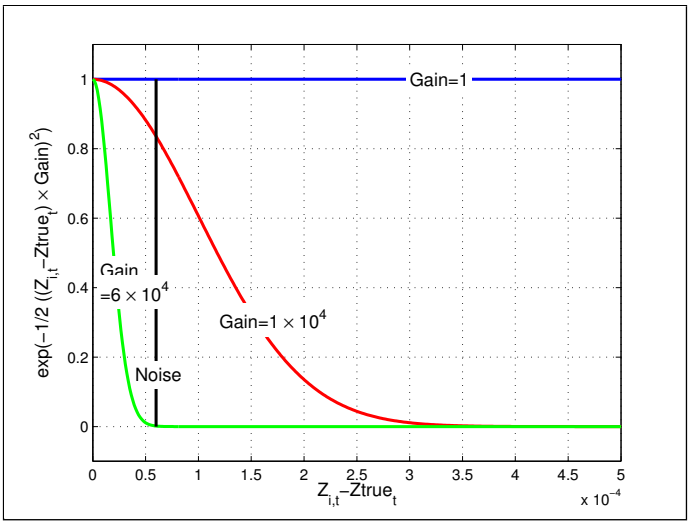

Figure 2.46 Comparison of selected particle filter gain values with sensor noise.

the actual heating source location receive the highest weight ($Z_{i,t} - Ztrue_t$ close to zero in Figure 2.46) and those particles far from the actual heating source location receive a weight close to zero. The non-dimensional sensor noise measured in the experiment above is approximately 6×10^{-5}. Therefore, the applied gain should yield the largest weights for $Z_{i,t} - Ztrue_t$ magnitudes between 0 and the noise of 6×10^{-5} and should yield weights close to 0 for $Z_{i,t} - Ztrue_t$ magnitudes larger than the sensor noise. Illustrated in Figure 2.46, a gain of 1×10^4 is too small to produce weights close to zero for $Z_{i,t} - Ztrue_t$ magnitudes larger than the sensor noise, but a gain of 6×10^4 precipitates the desired effect. Figure 2.47 illustrates the effect of the number of particles m on the convergence behavior and performance of the particle filter. The filter is quite robust and Figure 2.47 demonstrates the filter's ability to converge to the correct location with only 10 particles. The particle filter used in the comparisons with the other localization methods in the next section is based on 40 particles.

2.4.3.4 Least squares with ultrasonic pulse one-way time of flight measurement model

The ordinary least squares method is sometimes called the Gauss method of minimization [Woodbury, 2003a]. For the current localization, the estimated time of flight G for each sensor pair for a particular time t, a 4×1 matrix, depends on a vector of two unknowns in the state X and the value of G at $X = X + \Delta X$ is obtained through the truncated Taylor's series as

$$G|_{X+\Delta X} \approx G|_X + \left.\frac{\partial G}{\partial X}\right|_X \Delta X. \tag{2.48}$$

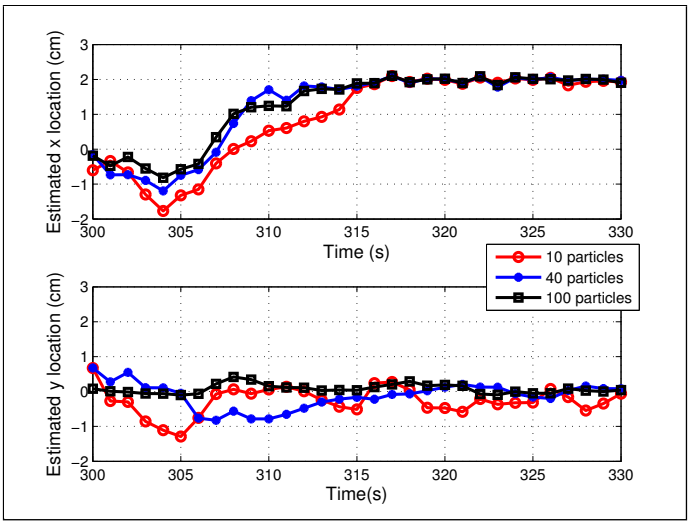

Figure 2.47 Particle filter convergence for selected numbers of particles with heating source located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$) and an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

The derivative in equation 2.48 is the 4×2 sensitivity matrix

$$B_t = \frac{\partial G}{\partial X} = \begin{bmatrix} \frac{\partial \overline{G}_1}{\partial x_1} & \frac{\partial \overline{G}_1}{\partial y_1} \\ \frac{\partial \overline{G}_2}{\partial x_2} & \frac{\partial \overline{G}_2}{\partial y_2} \\ \frac{\partial \overline{G}_3}{\partial x_3} & \frac{\partial \overline{G}_3}{\partial y_3} \\ \frac{\partial \overline{G}_4}{\partial x_4} & \frac{\partial \overline{G}_4}{\partial y_4} \end{bmatrix} \tag{2.49}$$

where t is time in seconds with a time step of 1 s, $\overline{G}_i$ with $i = 1,2,3,4$ is the ultrasonic pulse time of flight with the heating source located at (x_s, y_s), and (x_i, y_i) with $i = 1,2,3,4$ are the locations of four transducers. The experimentally obtained time of flight measurements are designated as Z_t, a 4×1 matrix. The desire is to improve the estimate for the state X_t based on the observations Z_t. The ordinary least squares objective function is

$$S = (Z_t - G|_{X_t} - B_t \Delta X_t)^T (Z_t - G|_{X_t} - B_t \Delta X). \tag{2.50}$$

The minimizer of equation 2.50 is found by forcing to zero the derivative with respect to ΔX resulting in the estimator

$$\Delta X_t = (B_t^{\mathrm{T}} B_t)^{-1} B_t^{\mathrm{T}} (Z_t - G_t|_{X_t}). \tag{2.51}$$

This method consists of obtaining expected temperatures from COMSOL®, computing the

average temperature between the transducers, and then computing an expected time of flight from equation 3.4. For the current analysis, the average temperature is computed along the path on the non-heated plate surface between the two sensors. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience since this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the x and y positions of all sensors by 0.0001 m.

2.4.3.5 Least squares with ellipse from ultrasonic pulse one-way time of flight measurement model

This method uses the same ellipse model detailed above with the extended Kalman filter and employs least squares method detailed above to locate the source. Figure 2.13 illustrates the geometry of an ellipse, where the two sensors are assumed to be the foci for the ellipse. Since the distance between sensors is known, ellipse parameters c and d can be related to each other and the ellipse can be represented with just one parameter c.

$$r_{is} + r_{js} = 2c = \sqrt{r_{ij}^2 + 4d^2} \tag{2.52}$$

where i and j are sensors and s is heat source.

$$c = \frac{1}{2}\sqrt{r_{ij}^2 + 4d^2} = \frac{r_{is} + r_{js}}{2} \tag{2.53}$$

$$r_{is} = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{2.54}$$

$$r_{js} = \sqrt{(x_j - x_s)^2 + (y_j - y_s)^2} \tag{2.55}$$

$$\frac{\partial c_i}{\partial x_s} = \frac{1}{2}\left[\frac{x_s - x_i}{r_{is}} + \frac{x_s - x_j}{r_{js}}\right] \tag{2.56}$$

$$\frac{\partial c_i}{\partial y_s} = \frac{1}{2}\left[\frac{y_s - y_i}{r_{is}} + \frac{y_s - y_j}{r_{js}}\right] \tag{2.57}$$

This measurement model consists of measuring the one-way ultrasonic pulse time of flight, using COMSOL® to obtain the average temperature between the transducers for a range of c values, using equation 3.4 to compute time of flight for the range of c values, and then interpolating the time of flight results using the spline method to obtain c for the measured time of flight. The

sensitivity matrix B_t is then

$$B_t = \begin{bmatrix} \frac{\partial c_1}{\partial x_s} & \frac{\partial c_1}{\partial y_s} \\ \frac{\partial c_2}{\partial x_s} & \frac{\partial c_2}{\partial y_s} \\ \frac{\partial c_3}{\partial x_s} & \frac{\partial c_3}{\partial y_s} \\ \frac{\partial c_4}{\partial x_s} & \frac{\partial c_4}{\partial y_s} \end{bmatrix}, \tag{2.58}$$

where t is time in seconds with a time step of 1 second, c_i with $i = 1,2,3,4$ is the ellipse parameter if the source is located at (x_s, y_s). The desire is to improve the estimate for the state X_t based on the observations c.

2.4.3.6 Results

Convergence behavior for the three inverse methods and both measurement models is compared in Figures 2.48 through 2.50. With the heating source located inside the sensor grid (Figure 2.48), the extended Kalman filter and the least squares with the direct model and the particle filter converge to the correct location while the extended Kalman filter and the least squares with the ellipse model do not. Similar convergence behavior is found with the heating source located at the edge of the sensor grid (Figure 2.49), although convergence is much faster. With the heating source located outside of the sensor grid (Figure 2.50), none of the methods converge to the correct location. These examples started with an initial guess of ($x = 0\,$cm, $y = 0\,$cm) for the heating source location.

Sensitivity to heating source location, explored in Section 2.3 and documented in previous work [Myers et al., 2012c], can help explain these results. With the heating source outside the sensor grid, only one sensor pair would have sufficient sensitivity to heating source location and only then if the heating source is located close to the sensor pair. With only one sensor pair receiving usable information, the inverse routine has insufficient information to converge on the correct heating source location. With the heating source located inside the sensor grid, all sensor pairs receive usable information and the inverse routine is able to converge to the correct location. With the heating source located at the edge of the sensor grid, one sensor pair receives usable information almost instantaneously resulting in faster convergence.

The extended Kalman filter with ellipse and least squares with ellipse models use the least amount of wall time and but the most memory of the five methods. Wall time for each iteration, independent of re- computing the COMSOL® model for the updated parameters, is approximately three times longer for the least squares time of flight model and almost four times longer for the extended Kalman filter time of flight model than the wall time for both of the ellipse models. The particle filter requires approximately 37 times more wall time than the ellipse models. Memory usage is lowest for the extended Kalman filter time of flight model and the least squares time of

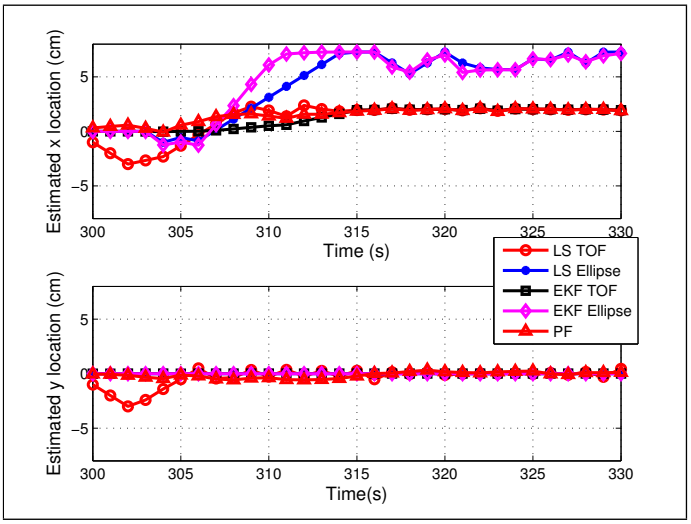

Figure 2.48 Least squares, extended Kalman filter, and particle filter convergence for both one-way ultrasonic pulse measurement models with the heating source located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$) and an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

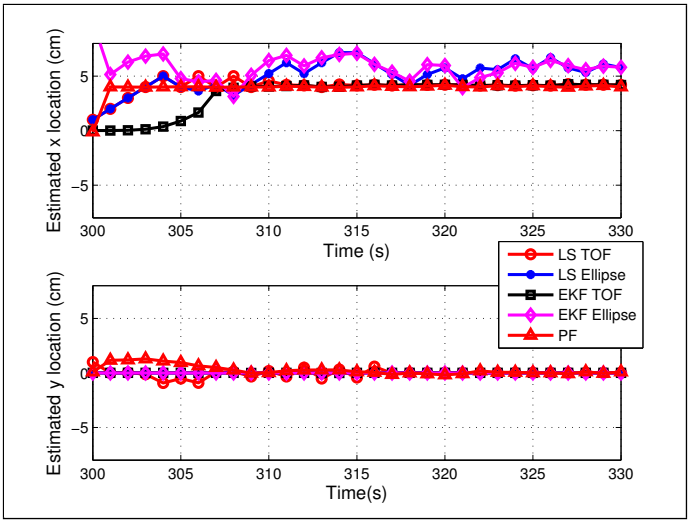

Figure 2.49 Least squares, extended Kalman filter, and particle filter convergence for both one-way ultrasonic pulse measurement models with the heating source located at ($x = 4\,\text{cm}$, $y = 0\,\text{cm}$) and an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

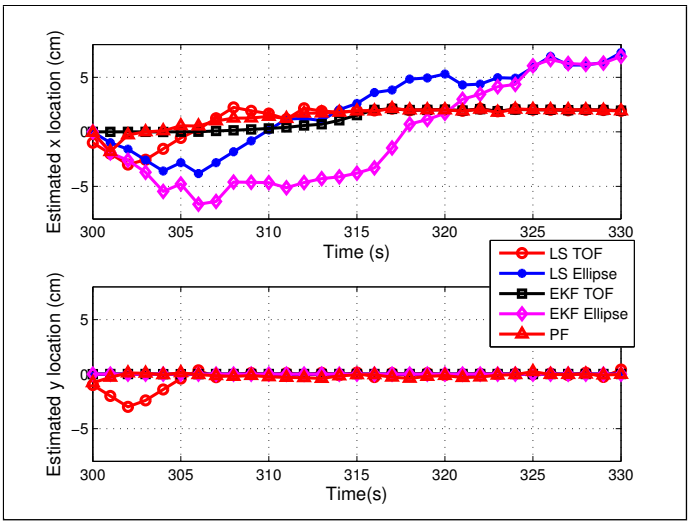

Figure 2.50 Least squares, extended Kalman filter, and particle filter convergence for both one-way ultrasonic pulse measurement models with the heating source located at ($x = 6\,\text{cm}$, $y = 0\,\text{cm}$) and an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

flight model. The particle filter requires approximately 30% more memory and the ellipse models require approximately four times more memory than the time of flight models.

Repeating the experiment using multiplexing equipment and a complete sensor grid of four transducers instead of simulating the sensor grid with separate experiments using two transducers might produce different convergence behavior, especially for heating source locations outside the sensor grid. While the experiment is reproducible, simulating a sensor grid with separate experiments introduces uncertainties that could effect the results. Additionally, sensor and heating source placement introduce uncertainties when simulating the sensor grid.

CHAPTER III

STEP SOURCE PARAMETER ESTIMATION

This chapter builds upon the findings from the spot heating source analysis in Chapter II by examining a step heating source that more closely resembles the heating profile of the boundary layer transition region on a hypersonic vehicle. Analysis in Chapter II sufficiently covered the comparison of inverse methods and measurement models. This chapter details analysis of the step heating source including sensitivity analysis, a thermocouple experiment, sensor array design, and heating source parameter estimation.

3.1 Sensitivity to source position, boundary conditions, and thermal conductivity

Similar to the sensitivity section for the spot heating source in Chapter II, this section details sensitivity analysis using the one-way ultrasonic pulse method.

3.1.1 Flat plate experiment

Consider a 30.5 cm x 30.5 cm x 0.2667 cm polished stainless steel 316L plate (Figure 3.1) with constant properties (Table 2.1). Flat black Rust-oleum® Specialty High Heat enamel paint is applied to a 19 cm $\times$ 14 cm rectangular area at the plate center to maximize energy absorption from the heater. Eight K-type 30 gauge thermocouples are attached on the non-heated side of the plate. With plate center being the origin and the x-axis being the length (Figure 3.1), thermocouples are attached at (x,y) locations of (0 cm, -2 cm), (0 cm, -5 cm), (0 cm, -8 cm), (0 cm, -12 cm), (-5 cm, -2 cm), (-5 cm, -5 cm), (-5 cm, -8 cm), and (-5 cm, -12 cm). The thermocouples are lightly dipped in thermal grease and secured to the plate with Kapton® tape to ensure good thermal contact. The heating source, The Designer's® Edge L-18 portable work light with 500 W halogen bulb, is positioned approximately 5 mm from the plate surface such that its beam strikes the black painted area and is applied at $t = 100$ s and removed at $t = 200$ s. The plate is oriented vertically with the positive y-axis pointing up.

3.1.2 3D conduction solution

The forward conduction solution used in this study is similar to the solution developed in Chapter II and leverages COMSOL Multiphysics® by the COMSOL Group and MATLAB® by The Mathworks, Inc. The solution uses a finite element mesh with smaller elements near the heat

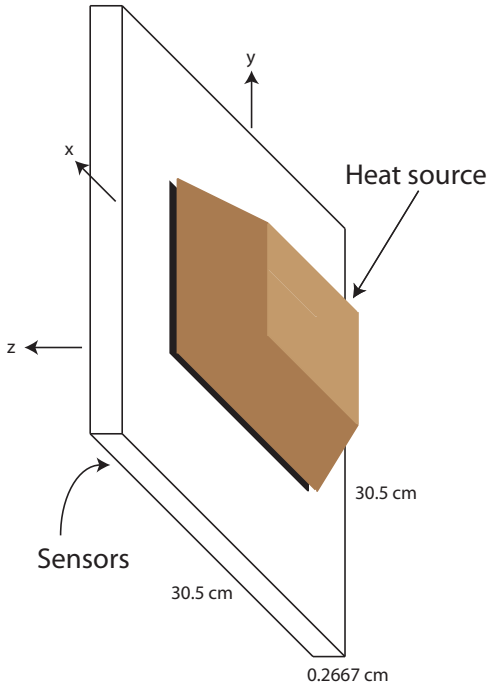

Figure 3.1 Illustration of flat plate with heat source and sensors (not drawn to scale).

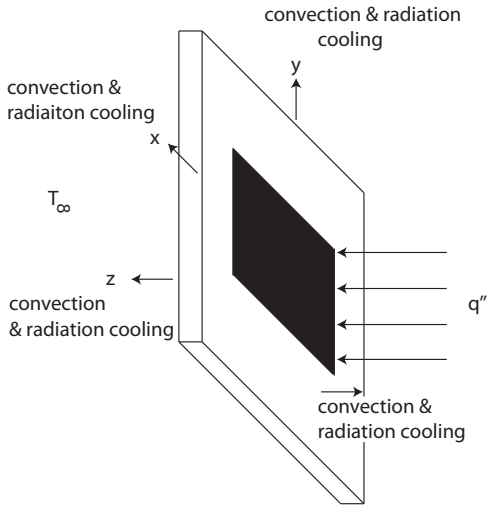

Figure 3.2 Illustration of boundary conditions on the flat plate.

source and larger elements near the plate edges to conserve computing resources.

For the flat plate detailed in Section 3.1.1, the governing equation for the subdomain (conduction in the plate) is

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = Q \tag{3.1}$$

where ∇ is the Laplacian and Q is an internal heat source (0 in this case). For the flat plate, k is assumed constant. Thus, the subdomain governing equation is

$$\nabla^2 T = \frac{\rho C_p}{k} \frac{\partial T}{\partial t} \tag{3.2}$$

where density (ρ), specific heat (C_p), and thermal conductivity (k) are considered constant. The boundary condition is

$$n \cdot (k \nabla T) = q_0 + h(T_{inf} - T) + \varepsilon \sigma (T_{amb}^4 - T^4) \tag{3.3}$$

where n is the surface normal vector, q_0 is the inward heat flux, h is the convection coefficient, ε is the surface emissivity, and σ is the Stefan-Boltzmann constant ($\sigma = 5.67 \times 10^{-8}\,\mathrm{W/m^2K^4}$). The initial condition is an isothermal plate at $T = 297.5\,\mathrm{K}$.

A grid convergence study is performed to ensure grid independence [Roache, 1998]. Both the number of elements in the plate's $x-y$ plane and the number of layers in the plate's thickness were considered. The grid convergence study led to the selection of a single mesh layer through the plate's thickness dimension, a maximum element size in the rectangular heated area of 0.1, 596 elements, and 3,759 degrees of freedom. Independent verification of the COMSOL® solution is performed using a closed-form, analytical solution of heating through a circular domain without convection or radiation [Kozlov et al., 1989]. Verification is performed using a COMSOL ® solution with a circular heated area instead of a rectangular heated area. The area of the heated zone for the circle model and the rectangle model were equal in magnitude. All other aspects of the models including maximum element size in the heated zone were identical. Agreement between the COMSOL solution and the closed-form solution is acceptable with mean absolute error less than 1.3 K. The maximum temperature rise is approximately 140 K, which occurs in the center of the heating source.

Parameter identification using least squares [Woodbury, 2003b] is used to determine the boundary conditions on the plate. Emissivity is $\varepsilon = 0.8$ and $\varepsilon = 0.075$ for the painted area and polished areas respectively [Doe, 2012]. Heat flux from the lamp is determined to be $q'' = 9.97\,\mathrm{KW/m^2}$ over the $19\,\mathrm{cm} \times 14\,\mathrm{cm}$ painted area and the convection coefficient on the plate sides to be $h = 12.9\,\mathrm{W/m^2\,K}$. This convection coefficient is larger than that determined for the spot source experiment (Chapter II, Section 2.1.3), however it falls within the normal range of

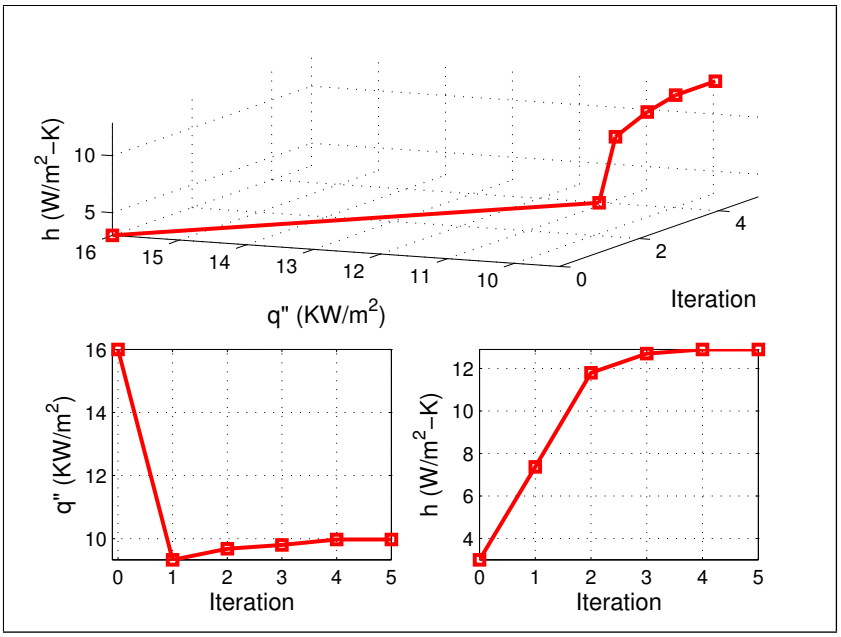

Figure 3.3 Least squares convergence for parameter identification.

$2-25\,\mathrm{W/m^2\,K}$ for free convection [Incropera and DeWitt, 2002]. The convection coefficient on the plate edges is assumed to be $h = 3\,\mathrm{W/m^2K}$. Figure 3.3 illustrates the convergence behavior for the parameter identification. Figures 3.4 and 3.6 illustrate the temperature response measured during the experiment with the temperature response of the model. The residuals [Beck and Woodbury, 1998, Dowding and Blackwell, 2001] are illustrated in Figures 3.5 and 3.7. Agreement between the model and the experiment is acceptable, however improvement could be achieved through modifications to the heating profile (e.g., accounting for non-uniformity in the heating source).

3.1.3 Measurement model

The measurement model examined in this work is based on a sensor array using four ultrasonic transducers in an 8 cm square pattern (Figure 3.8). Data acquisition equipment employing cross-correlation techniques would be used to determine and record ultrasonic one-way pulse time of flight readings once per second along the four paths shown in Figure 3.8.

The ultrasonic time of flight measurements are related to the average temperature between the transducers by [Myers et al., 2008, Myers et al., 2010c, Myers et al., 2010b, Myers et al., 2010a, Myers et al., 2012b]

$$G_{ij} = \frac{R_{ij}}{v_0}\left(1 + \xi\,\theta_{avg}\big|_i^j\right) \tag{3.4}$$

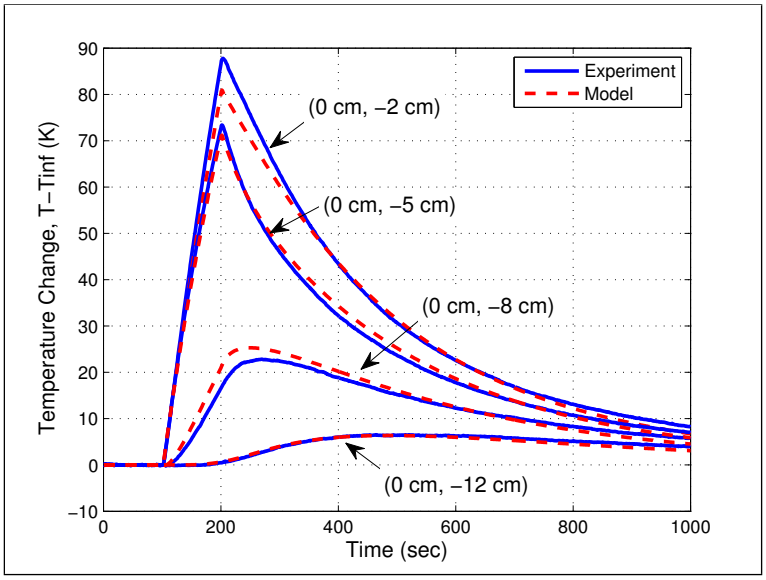

Figure 3.4 Comparison of the temperature response at four sensor locations along the plate centerline.

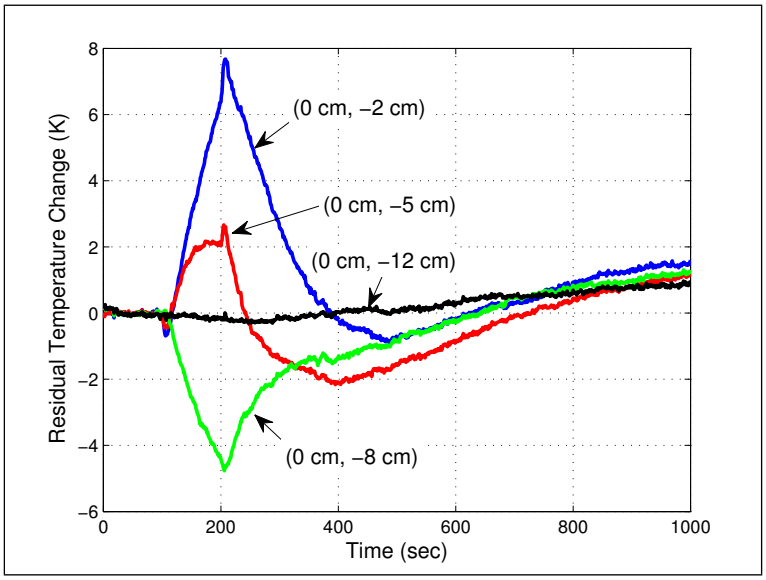

Figure 3.5 Residuals of the model when compared to the experiment measurements along the plate centerline.

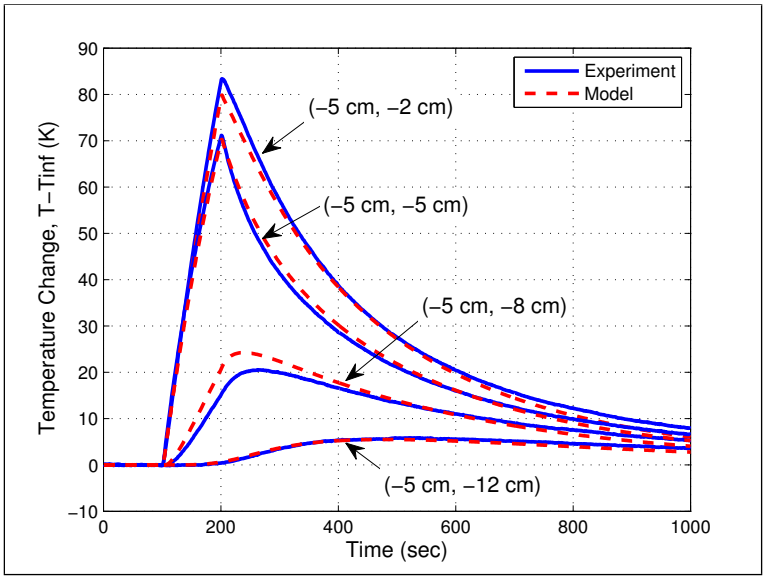

Figure 3.6 Comparison of the temperature response at four sensor locations along the plate offset line.

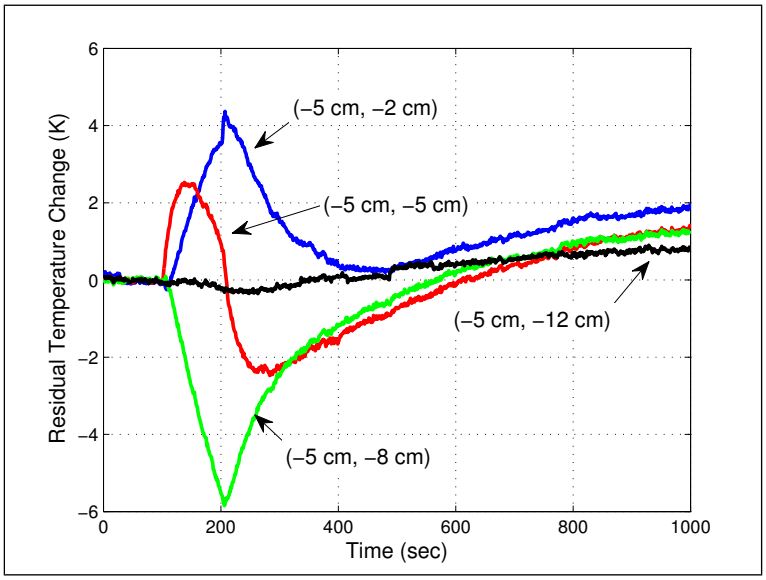

Figure 3.7 Residuals of the model when compared to the experiment measurements along the plate offset line.

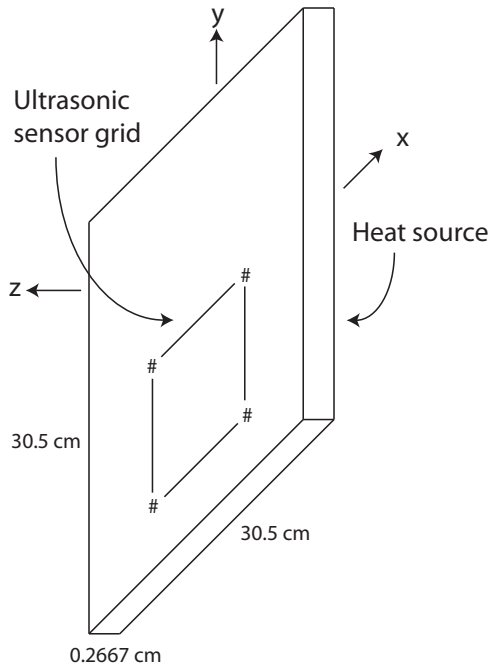

Figure 3.8 Ultrasonic sensor grid on the non-heated side of the plate with # symbols representing sensors and lines representing the ultrasonic pulse propagation paths between sensors (not drawn to scale).

where R_{ij} is the distance between transducers (m), v_0 is the sound speed in the material at a reference temperature, ξ is the ultrasonic time of flight factor, which is material dependent (Table 2.1), and θ_{avg} is the change in temperature from the reference temperature between the two sensors. Since R_{ij} is known with insufficient accuracy to compare the time of flight from the model to the measured values, the time of flight can be normalized to the initial state.

$$\frac{G_{ij}}{G_0} = \frac{\frac{R_{ij}}{v_0}\left[1+\xi\left(T_{avg}\big|_i^j - T_0\right)\right]}{\frac{R_{ij}}{v_0}} = 1+\xi\left(T_{avg}\big|_i^j - T_0\right) \tag{3.5}$$

where G_0 is the reference time of flight at T_0.

3.1.4 Sensitivity to source location

To better understand the four transducer array's sensitivity to source location, it is necessary to examine the sensitivity for one sensor pair. Note that this discussion assumes the source is static

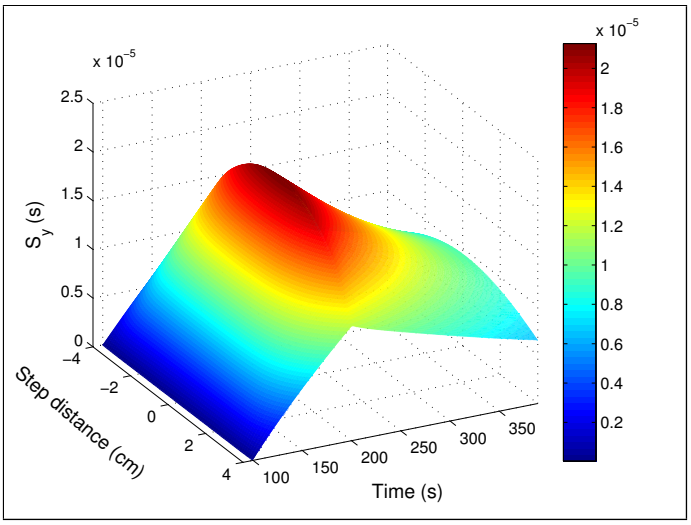

Figure 3.9 Heating source location sensitivity for a single sensor pair and the heating source step normal to the ultrasonic pulse propagation path. Heating is applied at $t = 100\,\text{s}$ and removed at $t = 200\,\text{s}$.

and therefore not moving with time. The sensitivity to heating source location is expressed as

$$S_y = y\frac{\partial G}{\partial y} \approx y\frac{G(x,y(1+\delta),z,t) - G(x,y,z,t)}{y(1+\delta) - y} \approx \frac{G(x,y(1+\delta),z,t) - G(x,y,z,t)}{\delta} \tag{3.6}$$

Figure 3.9 illustrates the sensitivity to heating source location when the step is normal to the ultrasonic pulse propagation path. Sensitivity is high when the step is between the sensors and drops off significantly when the step is outside of the sensors. Figure 3.10 illustrates the sensitivity to step location when the step is parallel to the ultrasonic pulse propagation path. Sensitivity is highest when the step is directly over the sensors at the end of the heating period. The magnitude of the sensitivity is higher over a larger area when the step is parallel than when the step is normal to the sensor path. This observation is important when considering sensor array design. For example, a designer would want to have more propagation paths parallel to the step.

3.1.5 Sensitivity to boundary conditions and thermal conductivity

Sensitivity to boundary conditions (Figure 3.2) and thermal conductivity are analyzed for an ultrasonic sensor configuration of four sensors in an 8 cm square configuration (Figure 3.8). Sensitivity for the primary heat flux (q''), convection coefficient for the plate sides (h), and thermal conductivity of the plate (k) are illustrated and analyzed for heating source step locations inside the sensor array and up to 2 cm outside the sensor array.

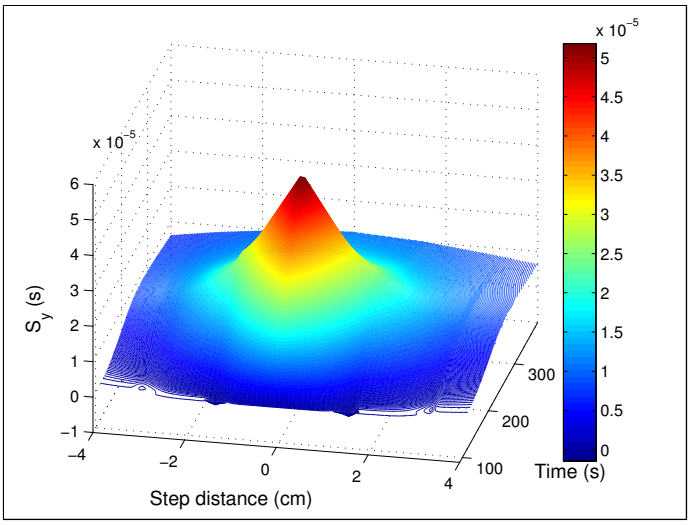

Figure 3.10 Heating source location sensitivity for a single sensor pair and the heating source step parallel to the ultrasonic pulse propagation path. Heating is applied at $t = 100\,\mathrm{s}$ and removed at $t = 200\,\mathrm{s}$.

Assumptions for the sensitivity to boundary conditions and thermal conductivity analysis are:

1. Source in fixed position (location unknown)
2. Source applied at time $t = 100\,\mathrm{s}$ and removed at $t = 200\,\mathrm{s}$
3. Source heat flux $q'' = 9.97\,\mathrm{KW/m^2}$ over $19\,\mathrm{cm} \times 14\,\mathrm{cm}$ rectangular area while source applied (value obtained above)
4. Convection coefficient $h = 12.9\,\mathrm{W/m^2K}$ on both sides of the plate (value obtained above)
5. Convection coefficient $h = 3\,\mathrm{W/m^2K}$ on the plate edges
6. Emissivity $\varepsilon = 0.8$ for the painted area (heated zone)
7. Emissivity $\varepsilon = 0.075$ for the non-heated area
8. Thermal conductivity $k = 14.6\,\mathrm{W/mK}$
9. Specific heat $C_p = 500\,\mathrm{J/kgK}$ and density $\rho = 8,000\,\mathrm{kg/m^3}$
10. Positions of sensors are $(\pm 4\,\mathrm{cm}, -3\,\mathrm{cm})$ and $(\pm 4\,\mathrm{cm}, -11\,\mathrm{cm})$ on the non-heated side

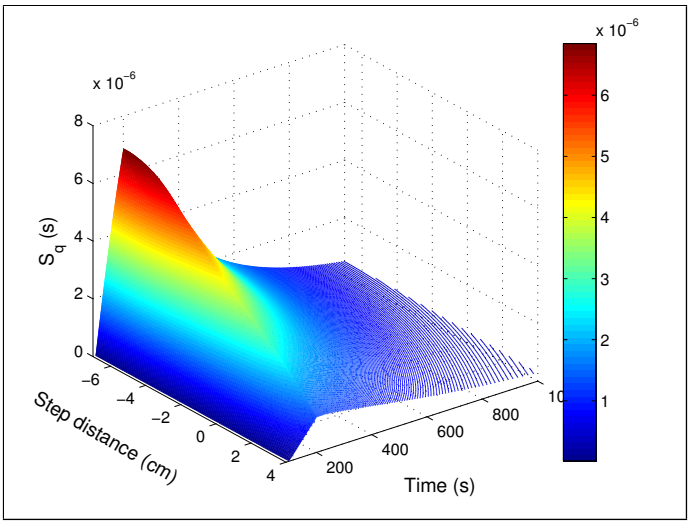

Figure 3.11 Scaled sensitivity to heat flux ($S_{q''}$) for a range of heating source step locations relative to sensor array center.

Sensitivity is computed using finite differences where baseline ultrasonic pulse time of flight values are computed using the assumptions above and then new ultrasonic pulse time of flight values are computed with the parameter being investigated multiplied by $1+\delta$. Sensitivities are presented scaled according to the relation [Beck and Arnold, 1977]

$$S_{\beta_i} = \beta_i \frac{\partial G}{\partial \beta_i} \tag{3.7}$$

$$S_{\beta_i} \approx \beta_i \frac{G(x,y,z,t,\beta_1,\cdots,\beta_i(1+\delta),\cdots,\beta_p) - G(x,y,z,t,\beta_1,\cdots,\beta_p)}{\beta_i(1+\delta)-\beta_i} \tag{3.8}$$

$$S_{\beta_i} \approx \frac{G(x,y,z,t,\beta_1,\cdots,\beta_i(1+\delta),\cdots,\beta_p) - G(x,y,z,t,\beta_1,\cdots,\beta_p)}{\delta} \tag{3.9}$$

where β_i is the parameter being investigated and G is ultrasonic pulse time of flight. The δ parameter used in this work is $\delta = 0.001$. By normalizing the sensitivities, direct comparison between all investigated parameters can be performed.

Figures 3.11, 3.12, and 3.13 illustrate sensitivity to heat flux (q''), convection coefficient (h), and thermal conductivity (k). As expected, the highest sensitivity to changes in the heat flux are when the heating source completely covers the sensor array since this arrangement requires the least amount of time for measurable heat to reach the sensors. Sensitivity is greatest for h and k when the heating source completely covers the sensor array.

Figure 3.14 illustrates the sensitivities for each parameter for all times during the experi-

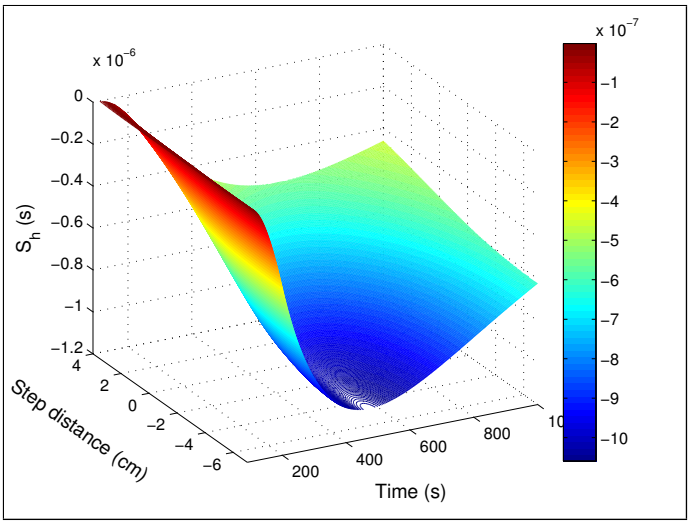

Figure 3.12 Scaled sensitivity to convection coefficient (S_h) for a range of heating source step locations relative to sensor array center.

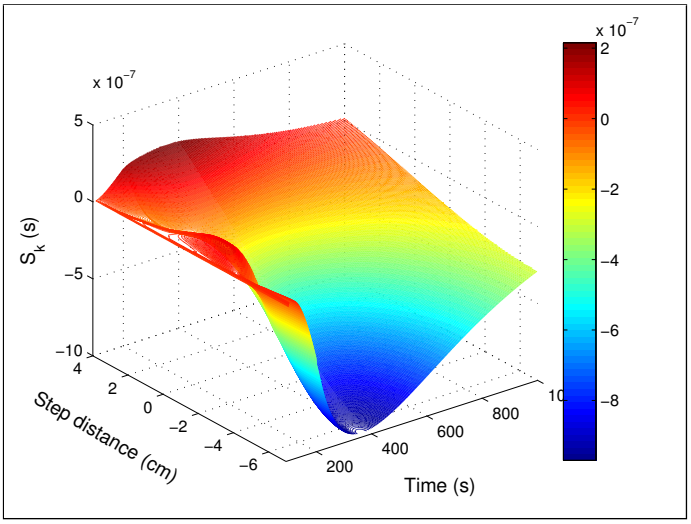

Figure 3.13 Scaled sensitivity to thermal conductivity (S_k) for a range of heating source step locations relative to sensor array center.

ment with the heating source in the center of the sensor array. Sensitivities to heat flux and thermal conductivity are correlated for heating portion of the experiment ($100\,\text{s} \leq t \leq 200\,\text{s}$) and stay correlated up to approximately $t = 250\,\text{s}$ making simultaneous estimation of q'' and k difficult. Figure 3.15 illustrates the sensitivities for each parameter for all times during the experiment with the heating source offset from the center of the sensor array by 2 cm which means less of the sensor array is covered by the heating source. The correlation between heat flux and thermal conductivity is not as strong. Figure 3.16 illustrates the sensitivities for each parameter for all times during the experiment with the heating source step offset from the center of the sensor array by $-2\,\text{cm}$, which means more of the sensor array is covered by the heating source. During heating, the correlation between heat flux and thermal conductivity is quite strong. Sensitivity to convection coefficient is not correlated with the other parameters, however during heating, the sensitivity magnitude is much smaller than the sensitivity magnitude of heat flux and thermal conductivity. Figure 3.17 illustrates the sensitivities for each parameter for all times during the experiment with the heating source offset from the center of the sensor array by 4 cm meaning that the step is located at the top edge of the sensor array. In this situation, only the sensor pair at the top of the array receives much usable information and then only after heating has been removed. Figure 3.18 illustrates the sensitivities for each parameter for all times during the experiment with the heating source offset from the center of the sensor array by $-4\,\text{cm}$. At this point, the entire sensor array is covered by the heating source and the step is at the lower edge of the sensor array. Sensitivity to heat flux dominates the solution rendering simultaneous estimation of q'' and either of the other two parameters difficult.

3.2 Sensor array design

A design of experiments on sensor array configuration is conducted using the candidate configurations in Figure 3.19. The height of all candidate configurations is consistent at 8 cm. The extended Kalman filter and COMSOL® are used to locate a simulated step source. Convergence behavior for all configurations is compared in Figures 3.20-3.22. The solution is able to converge to the correct heating source step location for all sensor configurations. As work progresses to moving sources, using a sensor configuration that yields quick convergence is highly desired. The square sensor array with six propagation paths and the hexagon sensor array provide the best convergence behavior for the widest range of heating source step locations.

3.3 Step source parameter estimation

Locating and characterizing a heating source depends upon many factors such as heating source movements in time, heating source magnitude changes in time, and other transient behaviors

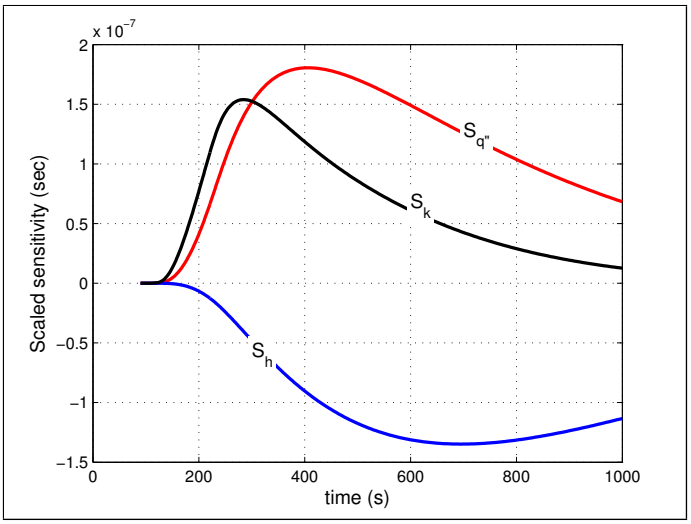

Figure 3.14 Scaled sensitivity for three parameters with heating source step located in the center of the sensor array. Heating is applied at $t = 100\,\mathrm{s}$ and removed at $t = 200\,\mathrm{s}$.

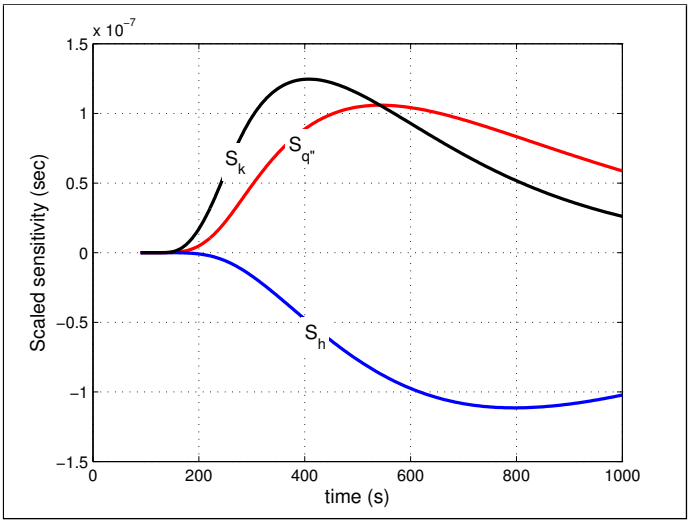

Figure 3.15 Scaled sensitivity for three parameters with the heating source step offset from the center of the sensor array by 2 cm. Heating is applied at $t = 100\,\mathrm{s}$ and removed at $t = 200\,\mathrm{s}$.

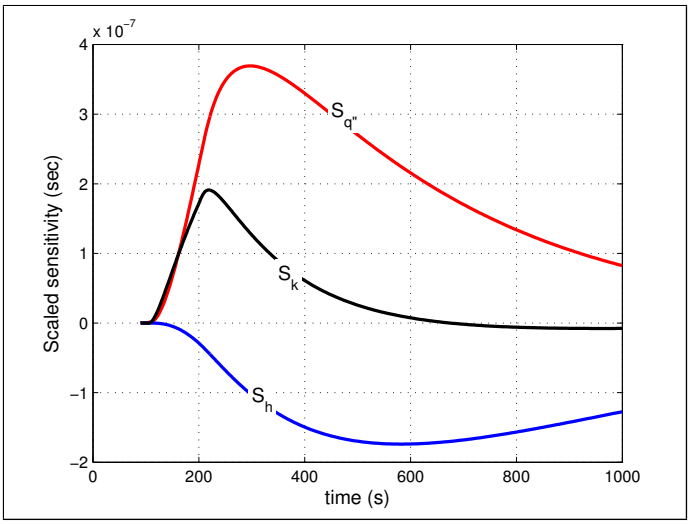

Figure 3.16 Scaled sensitivity for three parameters with the heating source step offset from the center of the sensor array by $-2\,\text{cm}$. Heating is applied at $t = 100\,\text{s}$ and removed at $t = 200\,\text{s}$.

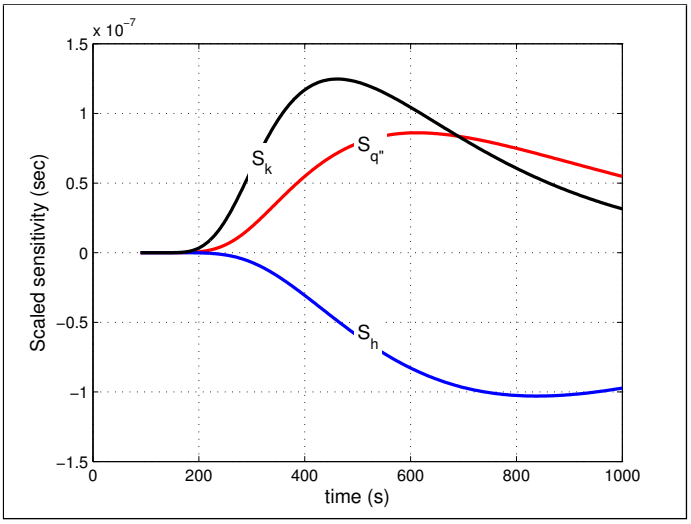

Figure 3.17 Scaled sensitivity for three parameters with the heating source step offset from the center of the sensor array by $4\,\text{cm}$. Heating is applied at $t = 100\,\text{s}$ and removed at $t = 200\,\text{s}$.

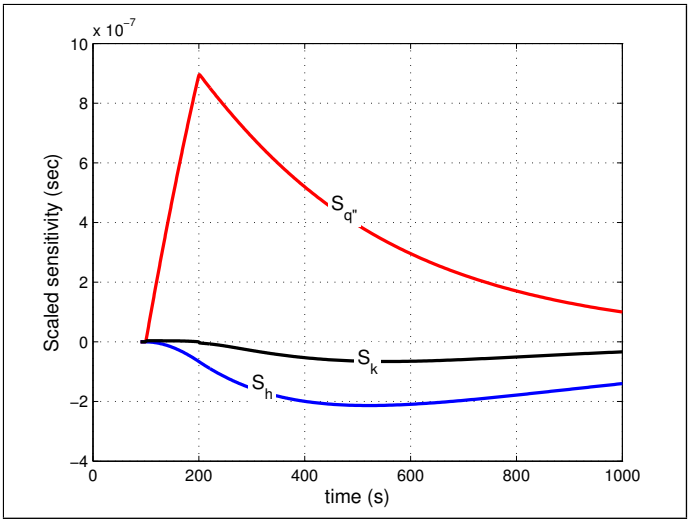

Figure 3.18 Scaled sensitivity for three parameters with the heating source step offset from the center of the sensor array by $-4\,\text{cm}$. Heating is applied at $t = 100\,\text{s}$ and removed at $t = 200\,\text{s}$.

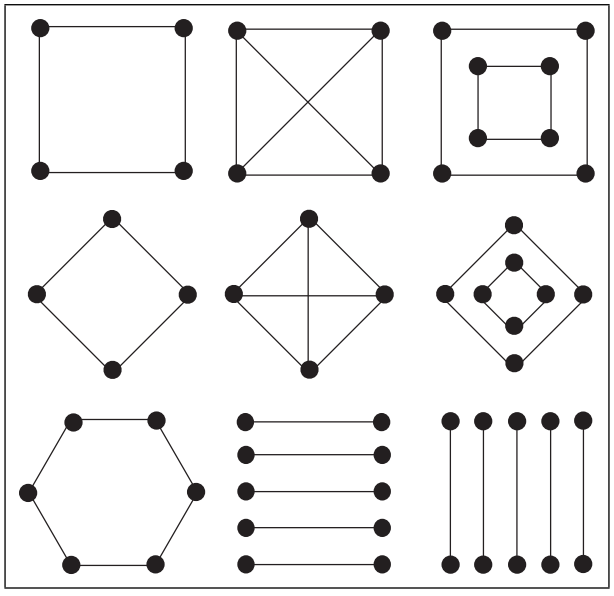

Figure 3.19 Design of experiments ultrasonic sensor array configurations. Each dot represents an ultrasonic transducer and the lines represent ultrasonic propagation paths.

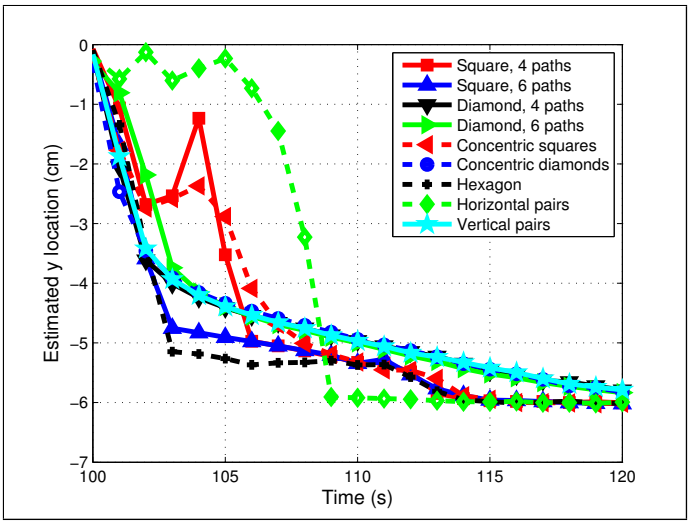

Figure 3.20 Extended Kalman filter convergence for sensor configurations in the design of experiments with the heating source step offset from the sensor array center by $-6\,\text{cm}$.

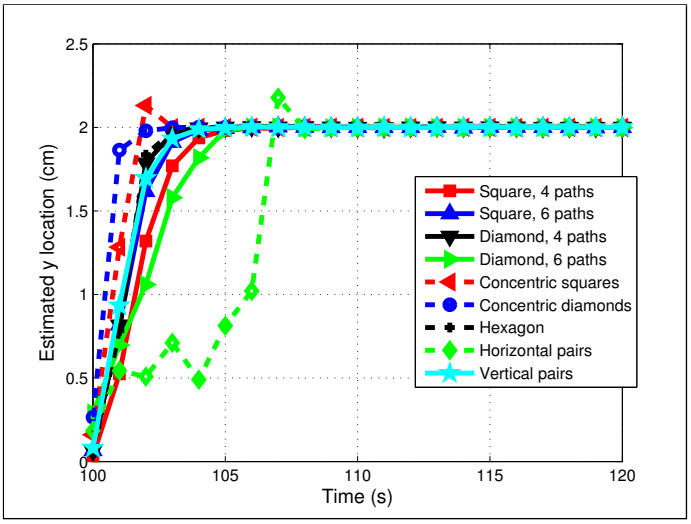

Figure 3.21 Extended Kalman filter convergence for sensor configurations in the design of experiments with the heating source step offset from the sensor array center by $2\,\text{cm}$.

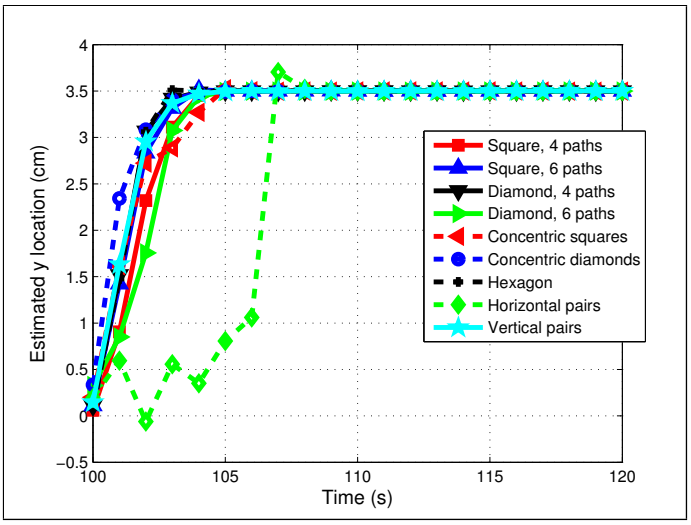

Figure 3.22 Extended Kalman filter convergence for sensor configurations in the design of experiments with the heating source step offset from the sensor array center by 3.5 cm.

(e.g., transient boundary conditions). Fairly restrictive assumptions can be imposed that simplify the problem. Analysis and algorithm development can proceed using these restrictive assumptions and then assumptions can be relaxed in stages to achieve the end result of source localization and characterization. The assumptions for this work are:

1. Source in fixed position (location unknown)
2. Source applied at time $t = 100\,\mathrm{s}$ and removed at $t = 200\,\mathrm{s}$
3. Source heat flux $q'' = 9.97\,\mathrm{KW/m^2}$ over $19\,\mathrm{cm} \times 14\,\mathrm{cm}$ rectangular area while source applied (value obtained above)
4. Convection coefficient $h = 12.9\,\mathrm{W/m^2K}$ on both sides of the plate (value obtained above)
5. Convection coefficient $h = 3\,\mathrm{W/m^2K}$ on the plate edges
6. Emissivity $\varepsilon = 0.8$ for the painted area (heated zone)
7. Emissivity $\varepsilon = 0.075$ for the non-heated area
8. Thermal conductivity $k = 14.6\,\mathrm{W/mK}$
9. Specific heat $C_p = 500\,\mathrm{J/kgK}$ and density $\rho = 8{,}000\,\mathrm{kg/m^3}$
10. Positions of sensors are ($\pm 4\,\mathrm{cm}$, $-3\,\mathrm{cm}$) and ($\pm 4\,\mathrm{cm}$, $-11\,\mathrm{cm}$) on the non-heated side

Table 3.1 Extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t(Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t)\overline{\Sigma}_t$
6	Return to Step 1 for next time step

11. Measurement and filter updates are performed in real-time with 1 s time steps.

The 1 s time step is based on the need for continuous, real-time parameter estimation for an operational system. Selection of the time step for system implementation will depend upon resource requirements and performance. The step source is a line located on the plate in the x direction. The state therefore is $X_t = y_q$. The ultrasonic time of flight is normalized by the time of flight before the heating source is applied to the plate (G_{ij}/G_0).

The extended Kalman filter algorithm to locate the source can be found in Table 3.2. There is no input (U_t) to the state; thus the extended Kalman filter state model is $a = 1$ and the state model Jacobian is $A = 1$. A state model variance of $\sigma^2 = 0.01\,\mathrm{m}^2$ is chosen as a baseline value. Thus, the state model covariance matrix is $Q_t = 0.01\,\mathrm{m}^2 \times I_2$.

This measurement model consists of obtaining expected temperatures from COMSOL®, computing the average temperature between the transducers, and then computing an expected time of flight to form $b(\overline{X}_t$ (equation 3.10). For the current analysis, the average temperature is computed along the path in the middle of the plate's thickness between the two sensors. The Jacobian partial derivatives are obtained using finite difference when moving the source in the y direction

(equation 3.11).

$$b(\overline{X}_t) = \begin{bmatrix} \overline{G}_1 \\ \overline{G}_2 \\ \overline{G}_3 \\ \overline{G}_4 \end{bmatrix}; \tag{3.10}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{G}_1}{\partial y_1} \\ -\frac{\partial \overline{G}_2}{\partial y_2} \\ -\frac{\partial \overline{G}_3}{\partial y_3} \\ -\frac{\partial \overline{G}_4}{\partial y_4} \end{bmatrix}, \tag{3.11}$$

where t is time in seconds with a time step of 1 s, $\overline{G}_i$ with $i = 1,2,3,4$ is the ultrasonic pulse time of flight with the heating source located at (y_s), and (y_i) with $i = 1,2,3,4$ are the locations of four transducers. The Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience because this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the y positions of all sensors by 0.0001 m. This value is based on a typical finite difference value of 0.1% multiplied by a unit length of 1 cm. Based on flat plate experiments in Section 2.4.1, the sensor noise is assumed be $\pm 6 \times 10^{-4}$ (a non-dimensional number based on G_{ij}/G_0) and is normally distributed ($\sigma^2 = ((6 \times 10^{-4})/3)^2 = 4 \times 10^{-7}$). The measurement covariance matrix, therefore, is $R = 4 \times 10^{-7} \times I_4$. Sensor position error is assumed to be normally distributed with a standard deviation of 1 mm in both x and y directions. Extended Kalman filter convergence behavior with these parameters is illustrated in Figure 3.23 for a range of step source locations and an initial guess of ($y = 0$ cm). The solution converges to the correct heating source step location in all cases. Similar convergence behavior is evident with all valid initial guesses (i.e., initial guesses physically on the plate).

Figure 3.24 illustrates the sensitivity to sensor noise for the one-way pulse time of flight measurement model. The experiment is simulated with the COMSOL® model using dimensionless noise comparable to the noise recorded during the experiments (6×10^{-4} s) and one and two orders of magnitude higher than the recorded noise (6×10^{-3} s and 6×10^{-2} s). The convergence behavior demonstrates the robustness inherent in the solution.

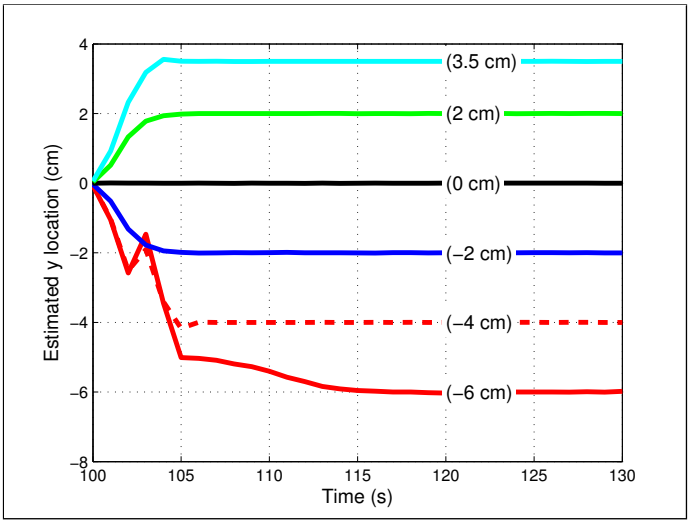

Figure 3.23 Extended Kalman filter convergence with the state model covariance values of $Q = 1 \times 10^{-2}\,\mathrm{m}^2$, measurement covariance values of $R = 4 \times 10^{-7} \times I_4$, heating step source located at a range of y values, and an initial guess of ($y = 0\,\mathrm{cm}$).

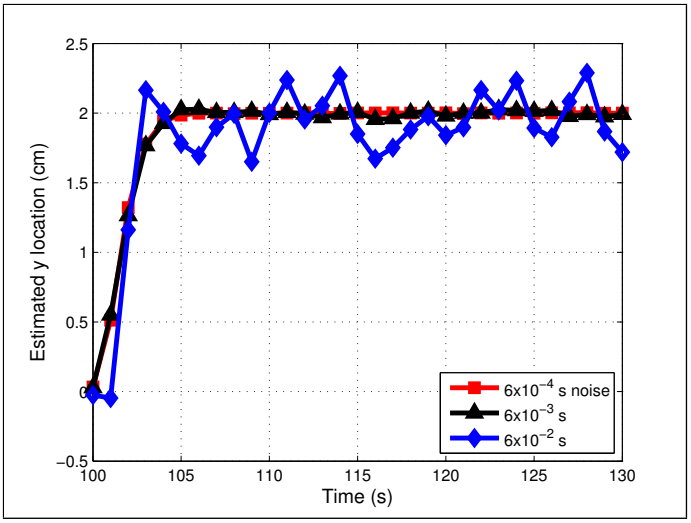

Figure 3.24 Extended Kalman filter convergence for a range of sensor noise values with the heating source step offset from the sensor array center by 2 cm and an initial guess of $y = 0\,\mathrm{cm}$.

3.4 Simultaneous Localization and Parameter identification (SLAP) of a moving step heating source

Previous sections imposed fairly restrictive assumptions to simplify the problem of locating and characterizing a step heating source. Factors such as heating source movements in time, heating source magnitude changes in time, and other transient behaviors (e.g., transient boundary conditions) were assumed static. This section relaxes these assumptions to consider a moving source [Myers et al., 2012e]. The square sensor array pattern with six propagation paths (Section 3.2) is used for this analysis.

3.4.1 Moving step source location estimation

The assumptions for this section are:

1. Source applied at time $t = 100\,\mathrm{s}$
2. Source heat flux $q'' = 9.97\,\mathrm{KW/m^2}$ over $19\,\mathrm{cm} \times 14\,\mathrm{cm}$ rectangular area while source applied (value obtained in Section 3.1.2)
3. Convection coefficient $h = 12.9\,\mathrm{W/m^2K}$ on both sides of the plate (value obtained in Section 3.1.2)
4. Convection coefficient $h = 3\,\mathrm{W/m^2K}$ on the plate edges
5. Emissivity $\varepsilon = 0.8$ for the painted area (heated zone)
6. Emissivity $\varepsilon = 0.075$ for the non-heated area
7. Thermal conductivity $k = 14.6\,\mathrm{W/mK}$
8. Specific heat $C_p = 500\,\mathrm{J/kgK}$ and density $\rho = 8{,}000\,\mathrm{kg/m^3}$
9. Positions of sensors are ($\pm 4\,\mathrm{cm}$, $-3\,\mathrm{cm}$) and ($\pm 4\,\mathrm{cm}$, $-11\,\mathrm{cm}$) on the non-heated side
10. Measurement and filter updates are performed in real-time with 1 s time steps.

The 1 s time step is based on the need for continuous, real-time parameter estimation for an operational system. Selection of the time step for system implementation will depend upon resource requirements and performance. The location of the step source is represented by a line located on the plate in the x direction. The state therefore is $X_t = y_q$. The ultrasonic time of flight is normalized by the time of flight before the heating source is applied to the plate (G_{ij}/G_0).

The extended Kalman filter algorithm to locate the source can be found in Table 3.2. There is no input (U_t) to the state; thus the extended Kalman filter state model is $a = 1$ and the state model

Table 3.2 Extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t (Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t) \overline{\Sigma}_t$
6	Return to Step 1 for next time step

Jacobian is $A = 1$. A state model variance of $\sigma^2 = 0.1\,\mathrm{m}^2$ is chosen as a baseline value. Thus, the state model covariance matrix is $Q_t = 0.1\,\mathrm{m}^2$.

This measurement model consists of obtaining expected temperatures from COMSOL®, computing the average temperature between the transducers, and then computing an expected time of flight to form $b(\overline{X}_t)$ (equation 3.12). For the current analysis, the average temperature is computed along the path in the middle of the plate's thickness between the two sensors. The Jacobian partial derivatives are obtained using finite differences when moving the source in the y direction (equation 3.13).

$$b(\overline{X}_t) = \begin{bmatrix} \overline{G}_1 \\ \overline{G}_2 \\ \overline{G}_3 \\ \overline{G}_4 \\ \overline{G}_5 \\ \overline{G}_6 \end{bmatrix}; \tag{3.12}$$

$$B_t = \begin{bmatrix} -\frac{\partial \overline{G}_1}{\partial y} \\ -\frac{\partial \overline{G}_2}{\partial y} \\ -\frac{\partial \overline{G}_3}{\partial y} \\ -\frac{\partial \overline{G}_4}{\partial y} \\ -\frac{\partial \overline{G}_5}{\partial y} \\ -\frac{\partial \overline{G}_6}{\partial y} \end{bmatrix}, \tag{3.13}$$

where t is time in seconds with a time step of 1 s, $\overline{G}_i$ with $i = 1,2,3,4,5,6$ is the ultrasonic pulse time of flight with the heating source located at (y_s), and (y) the location of the transducer. The

Jacobian B_t is constructed using the derivatives with respect to sensor position for convenience because this information can be obtained with one COMSOL® simulation. The derivatives are obtained from COMSOL® using finite differences by independently varying the y positions of all sensors by 0.0001 m. This value is based on a typical finite difference value of 0.1% multiplied by a unit length of 1 cm. Based on flat plate experiments in Section 2.4.1, the sensor noise is assumed be $\pm 6 \times 10^{-4}$ (a non-dimensional number based on G_{ij}/G_0) and is normally distributed ($\sigma^2 = ((6 \times 10^{-4})/3)^2 = 4 \times 10^{-7}$). The measurement covariance matrix, therefore, is $R = 4 \times 10^{-7} \times I_6$. Sensor position error is assumed to be normally distributed with a standard deviation of 1 mm in both x and y directions.

To accommodate a moving source, the COMSOL® solution is computed for the current time step with the step source at the estimated location. The temperature distribution from this solution is used as the initial condition for the next time step. A new temperature distribution is computed for the next time step based on a new estimate of the step source location. The plate is re-meshed for each time step.

Extended Kalman filter estimation behavior is illustrated in Figure 3.25 for the step source moving in a sine wave pattern and an initial guess of $y = 0$ cm (sensor array center). Heat flux magnitude q'' is known. The solution tracks the source fairly well but overshoots as the actual source's movement changes direction. Physical constraints of the plate dictated limits of possible step source locations to $-6\,\text{cm} < y < 3.7\,\text{cm}$ causing the flat portions of the estimated curves. Figures 3.26 and 3.27 illustrate the estimation behavior with an initial guess of $y = 2$ cm and $y = -2$ cm respectively. Estimation behavior is affected little from the initial guess of $y = 2$ cm since the actual source moves toward the initial guess. On the other hand, estimation behavior is affected more from the initial guess of $y = -2$ cm since the actual source moves away from the initial guess and the estimate must catch up and the overshoot is more pronounced.

Figures 3.28 and 3.29 illustrate estimation behavior when heat flux magnitude q'' is underestimated and overestimated respectively. The solution tracks the source in these cases, however the overshoot is more significant than with the case where the estimated heat flux matches the actual heat flux. Figure 3.30 illustrates estimation behavior for the step source moving in a sawtooth pattern. Tracking and overshoot for the sawtooth pattern is similar to that found for the sine wave pattern.

3.4.2 Step source heat flux estimation

Section 3.4.1 relaxed prior assumptions to consider a moving step source. One requirement for the location estimation is knowledge of heat flux magnitude q''. This section uses the same assumptions as Section 3.4.1 except now the heat flux magnitude q'' is unknown and the step source

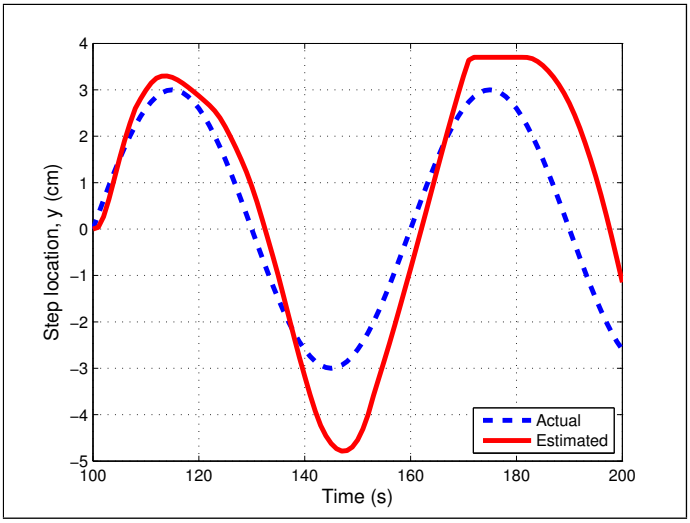

Figure 3.25 Extended Kalman filter estimation behavior for a step source moving in a sine wave pattern and an initial guess of $y = 0\,\text{cm}$ (sensor array center). Actual heat flux magnitude is known at $q'' = 9.97\,\text{KW}/\text{m}^2$.

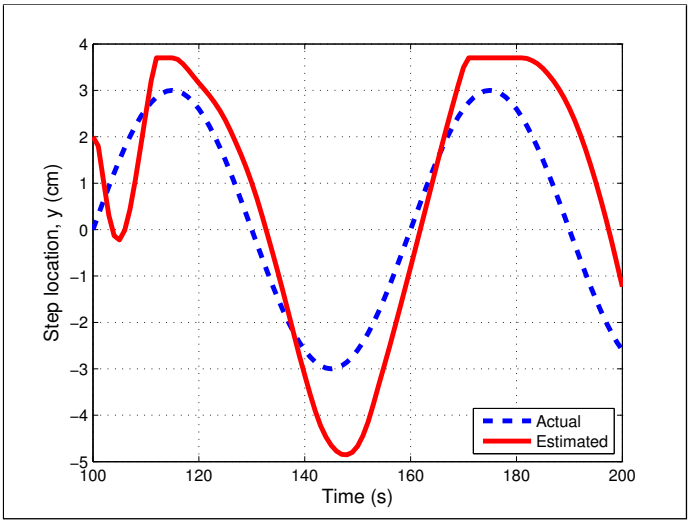

Figure 3.26 Extended Kalman filter estimation behavior for a step source moving in a sine wave pattern and an initial guess of $y = 2\,\text{cm}$. Actual heat flux magnitude is known at $q'' = 9.97\,\text{KW}/\text{m}^2$.

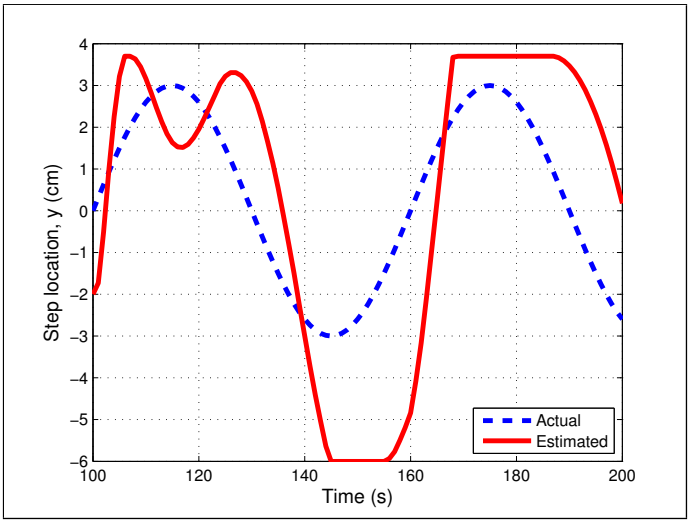

Figure 3.27 Extended Kalman filter estimation behavior for a step source moving in a sine wave pattern and an initial guess of $y = -2\,\text{cm}$. Actual heat flux magnitude is known at $q'' = 9.97\,\text{KW}/\text{m}^2$.

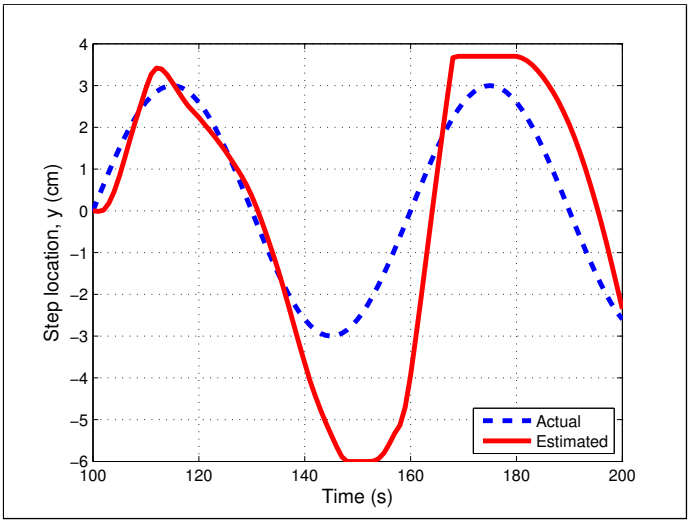

Figure 3.28 Extended Kalman filter estimation behavior for a step source moving in a sine wave pattern and an initial guess of $y = 0\,\text{cm}$. Heat flux magnitude is underestimated at 90% of the actual.

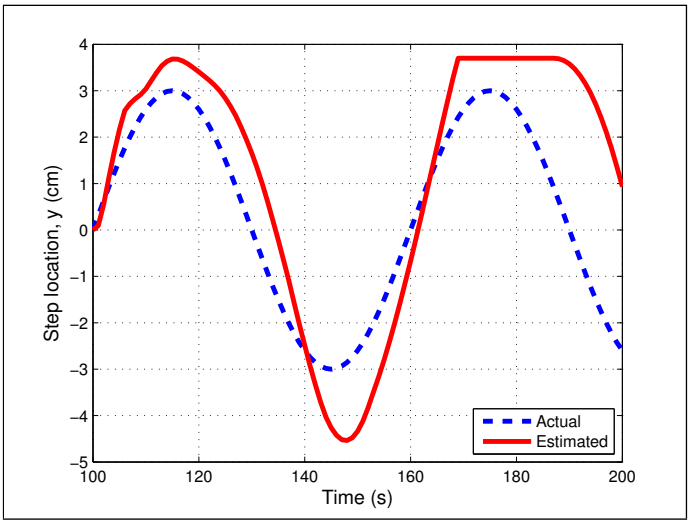

Figure 3.29 Extended Kalman filter estimation behavior for a step source moving in a sine wave pattern and an initial guess of $y = 0\,\text{cm}$. Heat flux magnitude is overestimated at 110% of the actual.

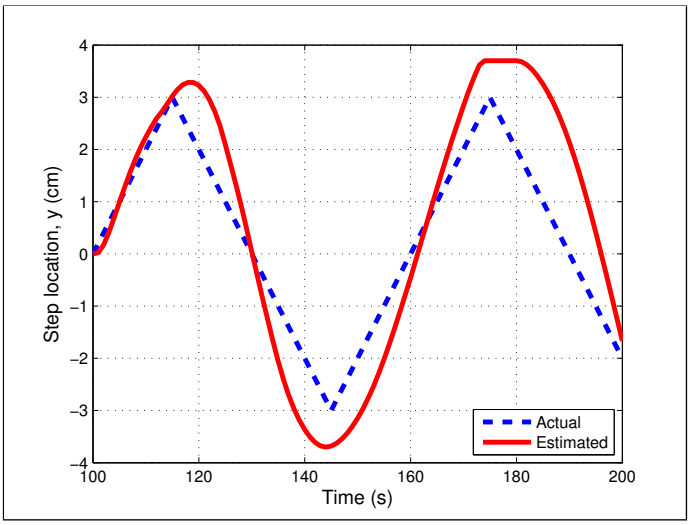

Figure 3.30 Extended Kalman filter estimation behavior for a step source moving in a sawtooth pattern and an initial guess of $y = 0\,\text{cm}$. Actual heat flux magnitude is known at $q'' = 9.97\,\text{KW}/\text{m}^2$.

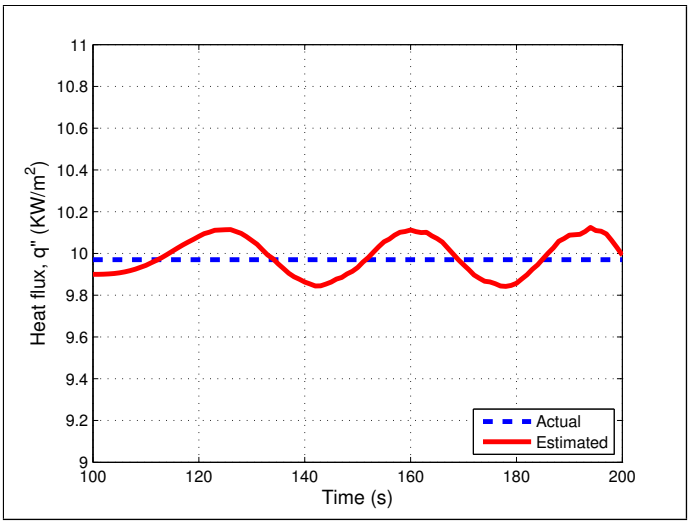

Figure 3.31 Extended Kalman filter estimation behavior for a stationary step source and an initial guess for heat flux magnitude of ($q'' = 9.0\,\mathrm{KW/m^2}$). 1 s time steps are used and step source location is known at $y = -4\,\mathrm{cm}$.

location is known and fixed at $y = -4\,\mathrm{cm}$. Figure 3.31 illustrates estimation behavior with an actual heat flux magnitude of $q'' = 9.97\,\mathrm{KW/m^2}$, an initial guess of $q'' = 9.0\,\mathrm{KW/m^2}$, and a 1 s time step. The estimation method oscillates near the actual magnitude with the maximum error less than 2%. Sensitivity analysis in Section 3.1.5 and documented in previous work determined sensitivity to heat flux magnitude is small immediately after the heating source is applied and increases over time [Myers et al., 2012d]. Therefore, a longer time step may produce better results. Figure 3.32 illustrates the sensitivity to heat flux for a range of step source positions. Heat flux sensitivity with the step source located in the center of the sensor array ($y = 0\,\mathrm{cm}$) remains near zero after 50 s of heating. The best case is with the step source located at the bottom of the sensor array ($y = -4\,\mathrm{cm}$). Figure 3.33 illustrates estimation behavior with a 10 s time step which produces slower oscillations, however no change is seen in the maximum error. Figure 3.34 illustrates estimation behavior with a variable heat flux magnitude and a 1 s time step. Maximum errors are greater than 20%.

Since the step source location procedure depends upon an accurate estimate of the heat flux magnitude, an alternative measurement model for heat flux may be warranted to augment the one-way ultrasonic pulse model for step source localization. A system of dual thermocouples could be used by drilling a hole in the aeroshell and installing one thermocouple flush with the exterior surface and the other thermocouple attached to the interior surface. A 1D forward conduction solution could then be used to recover heat flux at the aeroshell exterior surface. Instead, we propose installing one thermocouple on the interior surface collocated with selected transducers

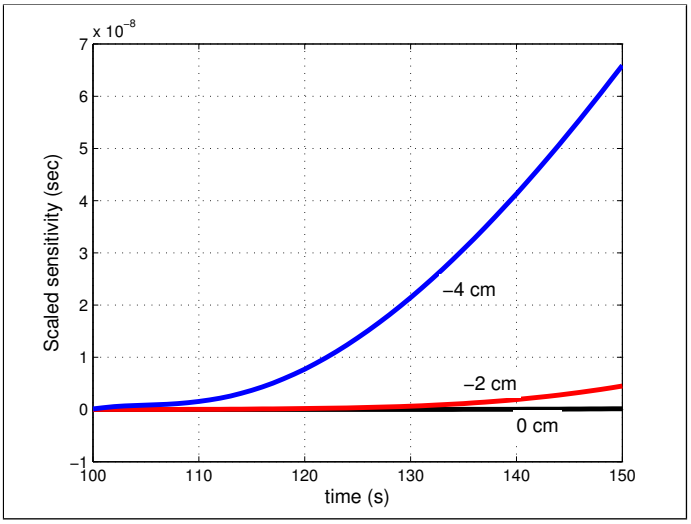

Figure 3.32 Scaled sensitivity to heat flux magnitude q'' for a range of step source positions after the heating source is applied.

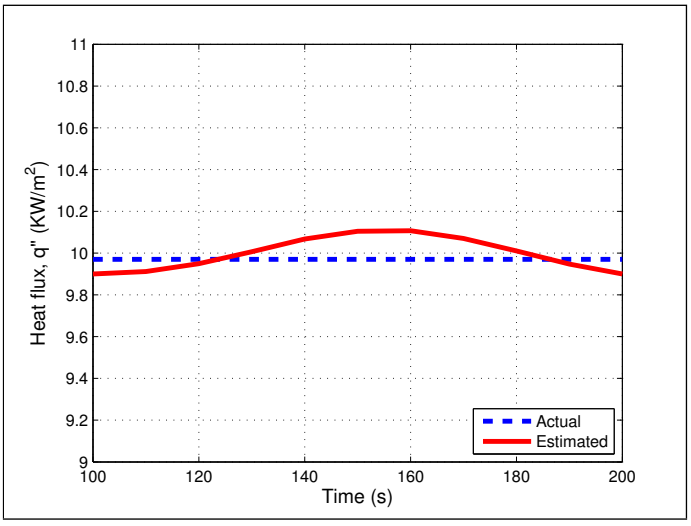

Figure 3.33 Extended Kalman filter estimation behavior for a stationary step source and an initial guess for heat flux magnitude of $q'' = 9.0\,\mathrm{KW/m^2}$. 10 s time steps are used and step source location is known at $y = -4\,\mathrm{cm}$.

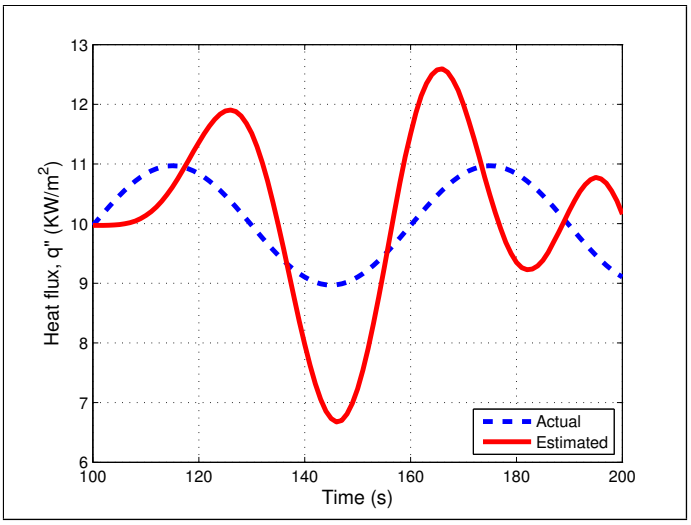

Figure 3.34 Extended Kalman filter estimation behavior for a stationary step source, a variable heat flux magnitude, and an initial guess for heat flux magnitude of ($q'' = 9.97\,\mathrm{KW/m^2}$). 1 s time steps are used and step source location is known at $y = -4\,\mathrm{cm}$.

and measuring the ultrasonic pulse-echo time of flight for a pulse transmitted through the thickness of the aeroshell, reflecting off the exterior boundary, and returning to the same transducer (Figure 1.3). Similar to Equation 1.1, time of flight for pulse-echo is related to temperature by

$$G = \frac{2L}{v_0} + \frac{\xi}{v_0}\int_0^L \theta(x)\,dx. \tag{3.14}$$

where L is the thickness of the medium, v_o is the sound speed at some reference temperature T_o, $\theta(x) = T(x) - T_o$ is the change in temperature relative to the reference, and ξ is the time of flight time temperature factor. The 2 accounts for the round trip time. Although the resulting integral still cannot be evaluated without knowledge of the temperature distribution, we recognize that the integral represents the total energy added to the system over time relative to a reference energy. For a control volume of a 1D wall with some heat transfer on both boundaries (Figure 3.35), an energy balance shows that

$$q''(x=0) = \rho c_p \int_0^L \frac{\partial \theta(x)}{\partial t}\,dx + q''(x=L). \tag{3.15}$$

For short time periods we can approximate the time derivative as a difference such that

$$q_0'' = \frac{\rho c_p}{\Delta t}\int_0^L \Delta\theta(x)\,dx + q_L'', \tag{3.16}$$

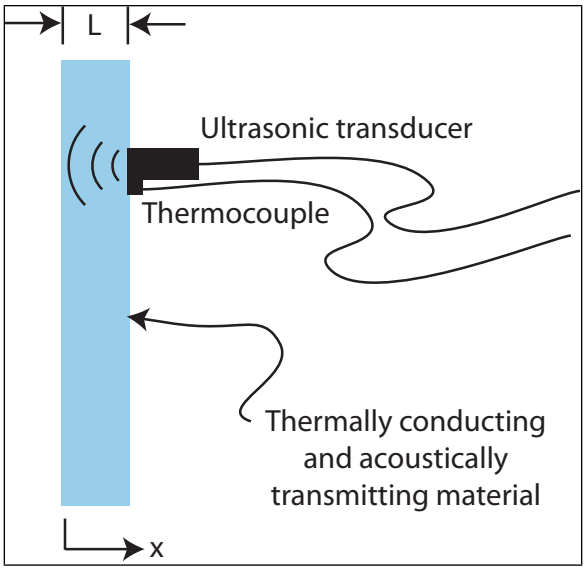

Figure 3.35 Ultrasonic pulse-echo technique.

where the subscripts 0 and L on q'' are used to identify the respective boundaries. The integral can be expressed in terms of ΔG obtained from equation 3.14, such that

$$q_0'' = \frac{\rho c_p}{\Delta t} \frac{v_0 \Delta G}{2\xi} + q_L''. \tag{3.17}$$

Equation 3.17 is significant because it relates an unknown and presumably inaccessible heat flux (q_0'') directly to a measurable quantity ΔG assuming we know what is happening on the accessible side (at $x = L$). Measuring accessible side heat flux q_L'' would require knowledge of accessible side thermal resistance and temperature differences between accessible side surface and the adjacent fluid or material.

CHAPTER IV

UNCERTAINTY AND EXTENDED KALMAN FILTER MODIFICATIONS

The previous chapters detail the extended Kalman filter algorithm and the need for defining the state model covariance and the measurement covariance. This chapter details an examination of the extended Kalman filter as implemented for the spot heating source parameter estimation problem detailed in Chapter II. The investigation encompasses sensitivity to changes in the state model covariance (Q), sensitivity to changes in the measurement covariance (R), the possibility that Q and R are correlated, behavior during convergence of the state covariance matrix (Σ), and behavior during convergence of the Kalman gain (K_t).

4.1 Extended Kalman filter observations

Figure 4.1 illustrates the sensitivity to the state model covariance by comparing values from $Q = 0.1\,\mathrm{m}^2 \times I_2$ to $0.000001\,\mathrm{m}^2 \times I_2$. Decreasing the state model covariance (Q) magnitude results in a damping effect on the convergence. Decreasing the magnitude too far causes the estimated position values to remain fairly constant and the solution fails to converge. Conversely, increasing the state model covariance (Q) magnitude increases the convergence rate. Increasing the state model covariance (Q) too far results in erratic position estimates and the solution fails to converge.

Figure 4.2 illustrates the sensitivity to the measurement covariance by comparing values from $R = 4 \times 10^{-5} \times I_4$ to $R = 4 \times 10^{-10} \times I_4$. Increasing the measurement covariance (R) magnitude results in a damping effect on the convergence. Increasing the magnitude too far causes the estimated position values to remain fairly constant and the solution fails to converge. Conversely, decreasing the measurement covariance (R) magnitude increases the convergence rate. Decreasing the measurement covariance (R) too far results in erratic position estimates and the solution fails to converge.

A trend is evident when comparing Figures 4.1 and 4.2 in that Q and R appear to be inversely correlated. Figure 4.3 illustrates the relationship. Decreasing Q by one order of magnitude or increasing R by one order of magnitude results in similar convergence behavior. Likewise, increasing Q by one order of magnitude or decreasing R by one order of magnitude also results in similar convergence behavior. For example, using $Q = 0.0001\,\mathrm{m}^2 \times I_2$ and $R = 4 \times 10^{-7} \times I_4$ as the baseline, decreasing the state model covariance to $Q = 0.00001\,\mathrm{m}^2 \times I_2$ but keeping the measurement covariance at $R = 4 \times 10^{-7} \times I_4$ results in similar convergence behavior if the state model covariance is kept at $Q = 0.0001\,\mathrm{m}^2 \times I_2$ and the measurement covariance is increased to

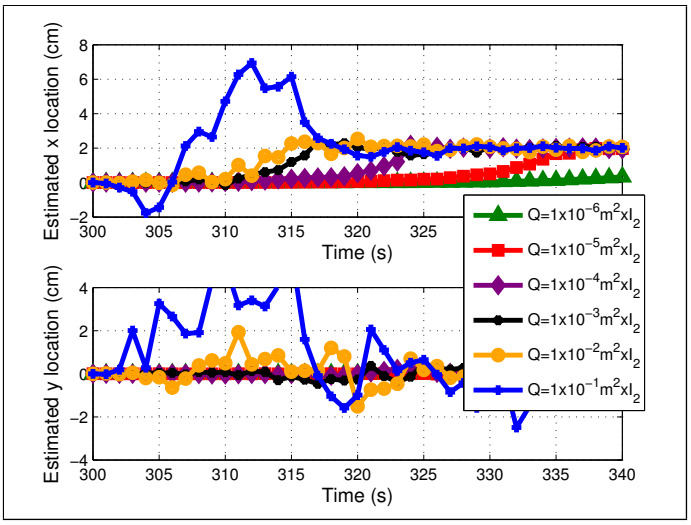

Figure 4.1 Extended Kalman filter convergence for a range of state model covariance values (Q) with constant measurement covariance values of $R = 4 \times 10^{-7} \times I_4$, the heating source located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$), and an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

$R = 4 \times 10^{-6} \times I_4$.

We can conclude from these observations that the state model covariance (Q) and the measurement covariance (R) are correlated for this heating source localization scenario. The measurement covariance is determined from sensor noise, a measurable quantity, and the state model covariance is unknown and not measurable. Therefore, a large uncertainty exists in the state model covariance while a small uncertainty exists in the measurement covariance. Selection of an appropriate state model covariance must then be obtained through a parameter sweep while observing convergence behavior.

4.2 Adaptive extended Kalman filter

The goal of exploring adaptations to the extended Kalman filter is to obtain faster and smoother convergence. Since the measurement covariance is known with a small uncertainty, the state model covariance is unknown, and the state model covariance and measurement covariance are correlated, an adaptive extended Kalman filter is envisioned where changes are made during each iteration to the state model covariance to improve convergence. A value is needed from the extended Kalman filter to drive changes to the state model covariance. Examining the extended Kalman filter (Table 4.1) reveals two possible sources: the Kalman gain (K_t) and the state covariance (Σ_t). The Kalman gain specifies the degree that the measurement update Z_t is incorporated

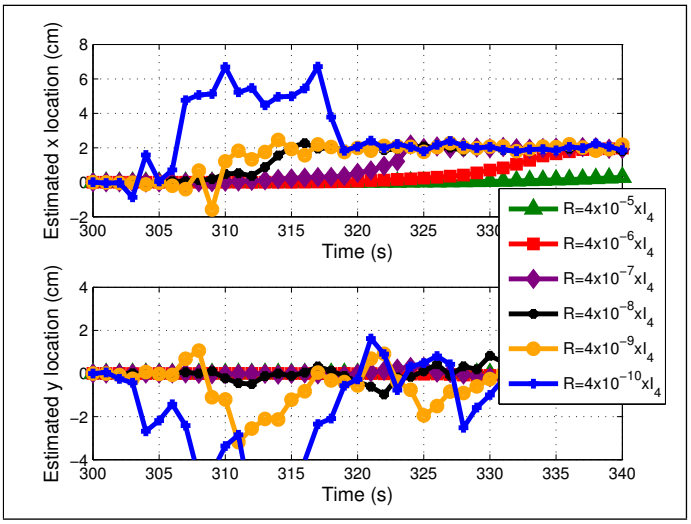

Figure 4.2 Extended Kalman filter convergence for a range of measurement covariance values (R) with constant state model covariance values of $Q = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$, the heating source located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$), and an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

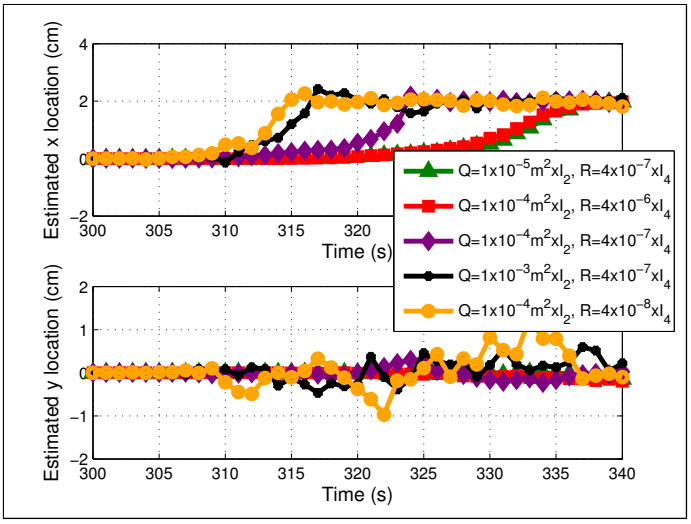

Figure 4.3 Extended Kalman filter convergence illustrating the correlation between the state model covariance matrix (Q) and the measurement covariance matrix (R). The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

Table 4.1 Extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t(Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t)\overline{\Sigma}_t$
6	Return to Step 1 for next time step

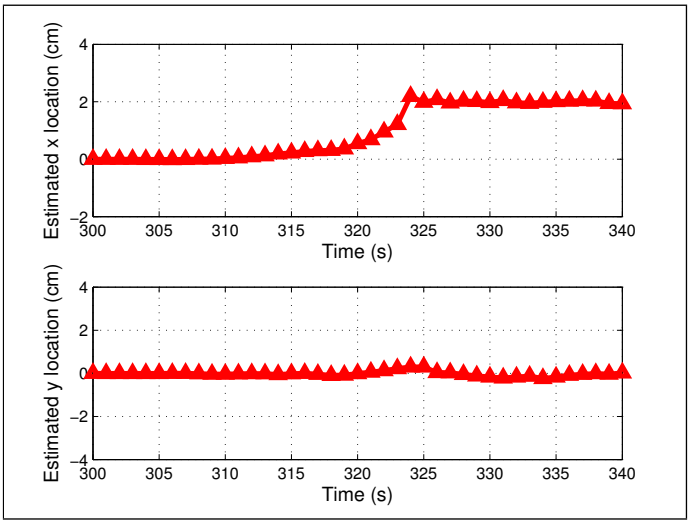

Figure 4.4 Extended Kalman filter convergence with the state model covariance values of $Q = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$, measurement covariance values of $R = 4 \times 10^{-7} \times I_4$, heating source located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$), and an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

into the new state estimate X_t while the state covariance Σ_t represents the uncertainty in the new state estimate X_t.

Figure 4.5 illustrates the magnitude of each element in the Kalman gain (K_t), which, for this heating source localization, is a 2×4 matrix. Comparing Figure 4.5 with Figure 4.4, the $(1,3)$ value from the Kalman gain (K_t) stands out as a possible source since it increases until convergence and then decreases rapidly. However, this value remains large even after convergence.

Figure 4.6 illustrates the normalized magnitude of the variance contained in the state covariance matrix (Σ) which is a 2×2 matrix in this heating source localization problem. The variance

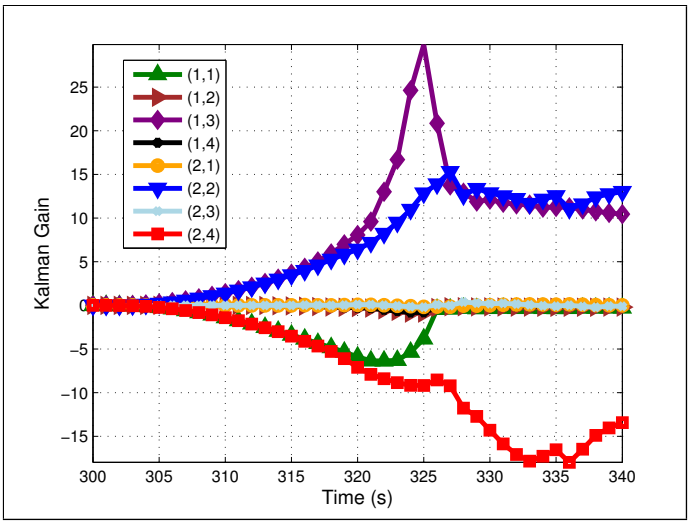

Figure 4.5 Kalman gain values during convergence for state model covariance of $Q = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$ and measurement covariance of $R = 4 \times 10^{-7} \times I_4$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$). Legend entries refer to the matrix element in the Kalman gain which is a 2×4 matrix. Convergence (from Figure 4.4) is at 324 s.

values are in the main diagonal of the matrix and are identical. Comparing with Figure 4.4, we observe that the variance increases steadily, decreases rapidly just before and during convergence, and remains small after convergence. Figure 4.7 illustrates the normalized magnitude of the variance for a range of state model covariance values from $Q = 1 \times 10^{-1}\,\mathrm{m}^2 \times I_2$ to $Q = 1 \times 10^{-6}\,\mathrm{m}^2 \times I_2$ and a measurement covariance of $R = 4 \times 10^{-7} \times I_4$. A comparison of Figure 4.7 with Figure 4.1 yields the observation that the variance increases steadily, decreases rapidly just before and during convergence, and remains small after convergence for every state model covariance examined.

Based on these observations, an adaptive extended Kalman filter is developed [Myers et al., 2012f] and is presented in Table 4.2 and illustrated in Figure 4.8. Step 6 is the only change from the extended Kalman filter found in Table 4.1. The state model covariance matrix Q is modified at the end of each iteration based on the state covariance Σ and the rate of change in the estimated state ΔX_t. Three conditions are possible when modifying Q. First, if the covariance is increasing at a rate greater than a predefined tolerance value and if the estimated state is changing less than a predefined limit, Q is multiplied by an predefined adaptive gain M. This adaptive gain will have a value greater than 1, which, in this first condition, has the effect of increasing the magnitude of Q and increasing the rate of convergence. From the analysis above, an increasing state covariance Σ indicates the solution is not converged and if the change in the estimate state is below a threshold,

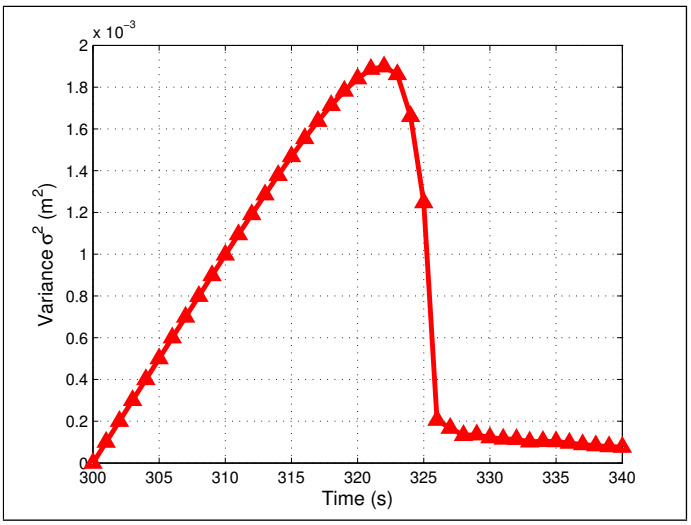

Figure 4.6 Variance (σ^2) from the state covariance matrix (Σ) for state model covariance of $Q = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$ and measurement covariance of $R = 4 \times 10^{-7} \times I_4$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$). Convergence (from Figure 4.4) is at 324 s.

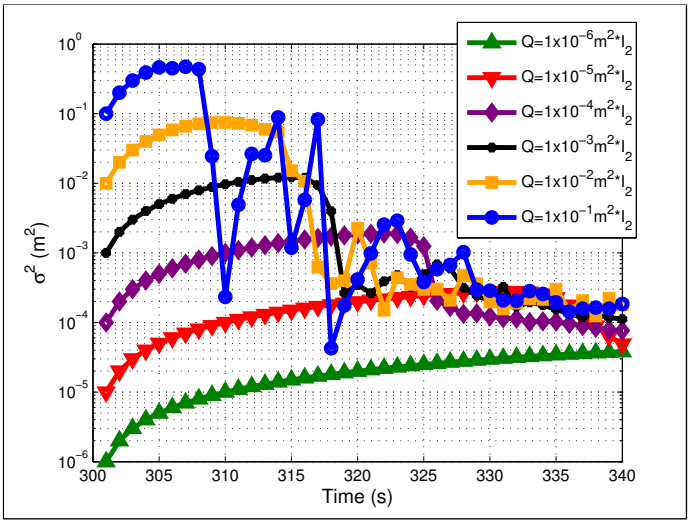

Figure 4.7 Variance (σ^2) from the state covariance matrix (Σ) for a range of state model covariance values (Q) and constant measurement covariance of $R = 4 \times 10^{-7} \times I_4$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$). Convergence behavior can be found in Figure 4.1.

Table 4.2 Adaptive extended Kalman filter algorithm.

Step	Operation
1	$\overline{X}_t = a(U_t, X_{t-1})$
2	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
3	$K_t = \overline{\Sigma}_t B_t^T (B_t \overline{\Sigma}_t B_t^T + R_t)^{-1}$
4	$X_t = \overline{X}_t + K_t(Z_t - b(\overline{X}_t))$
5	$\Sigma_t = (I - K_t B_t)\overline{\Sigma}_t$
6	$Q_{t+1} = \begin{cases} Q_t * M_t & \text{if } \lvert\Delta\Sigma_t\rvert > \Sigma_{tolerance} \text{ and } \Delta X_t < \Delta X_{limit} \\ Q_t / M_t & \text{if } \lvert\Delta\Sigma_t\rvert > \Sigma_{tolerance} \text{ and } \Delta X_t \geq \Delta X_{limit} \\ Q_t & \text{if } \lvert\Delta\Sigma_t\rvert \leq \Sigma_{tolerance} \end{cases}$
7	Return to Step 1 for next time step

convergence time can be reduced by increasing the magnitude of Q. Second, if the covariance is increasing at a rate greater than a predefined tolerance value and if the estimated state is changing more than a predefined limit, Q is divided by the predefined gain M. In this second condition, increasing the magnitude of Q might cause erratic convergence or a failure to reach a solution. Thus, by reducing the magnitude of Q, convergence is dampened. Third, if the covariance is increasing at a rate less than a predefined tolerance value, no change is needed to Q.

4.3 Results

The adaptive extended Kalman filter developed in this chapter incorporates three new parameters. The predefined tolerance value $\Sigma_{tolerance}$ for changes to the state covariance Σ is based on the variance found in the first iteration and is defined as $\Sigma_{tolerance} = \Sigma_1$. The magnitude of Σ is dependent upon filter parameters including the state model covariance Q, thus basing the tolerance on the first iteration ensures the adaptive nature of the filter will smooth convergence near the converged solution. The predefined limit to convergence rate ΔX_{limit} for this work is defined as $\Delta X_{limit} = [\Delta x_{limit}, \Delta y_{limit}]^T = [1\,\text{cm/sec}, 1\,\text{cm/sec}]^T$. Figure 4.9 illustrates convergence for the adaptive extended Kalman filter with $Q_0 = 1 \times 10^{-4}\,\text{m}^2 \times I_2$, $R = 4 \times 10^{-7} \times I_4$, and an adaptive gain of $M_t = 2$. The adaptive extended Kalman filter outperforms the extended Kalman filter in this example. Figure 4.10 illustrates the variance value in Q during convergence for the adaptive extended Kalman filter while Figure 4.11 illustrates the variance values from Σ for a range of initial state model covariance Q_0 values.

Figure 4.12 illustrates the effect that different initial state model covariance values has on the adaptive extended Kalman filter convergence. Comparing with Figure 4.1, the adaptive

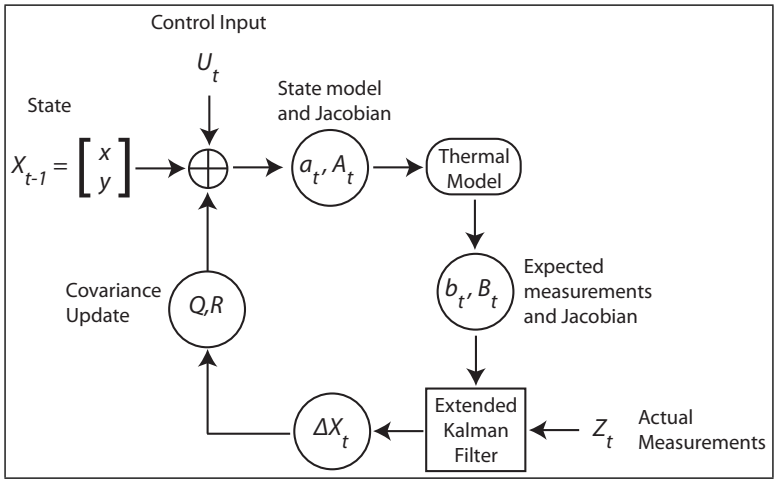

Figure 4.8 Parameter estimate update process for the adaptive extended Kalman filter and the COMSOL® model.

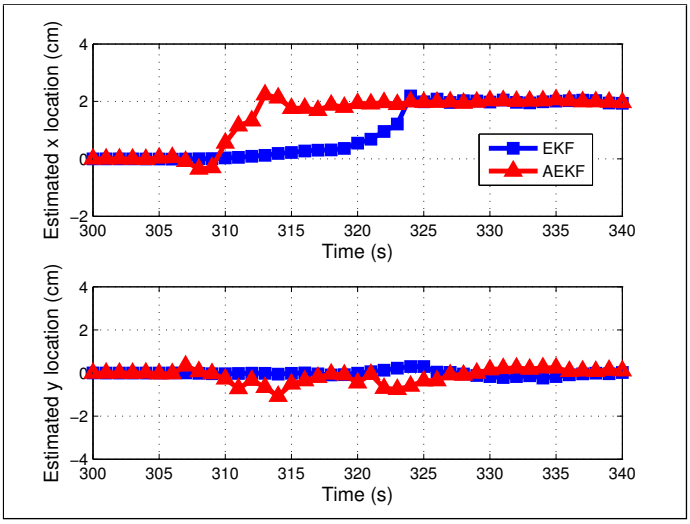

Figure 4.9 Extended Kalman filter and adaptive extended Kalman filter convergence with $Q_0 = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$, $R = 4 \times 10^{-7} \times I_4$, and $M = 2$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

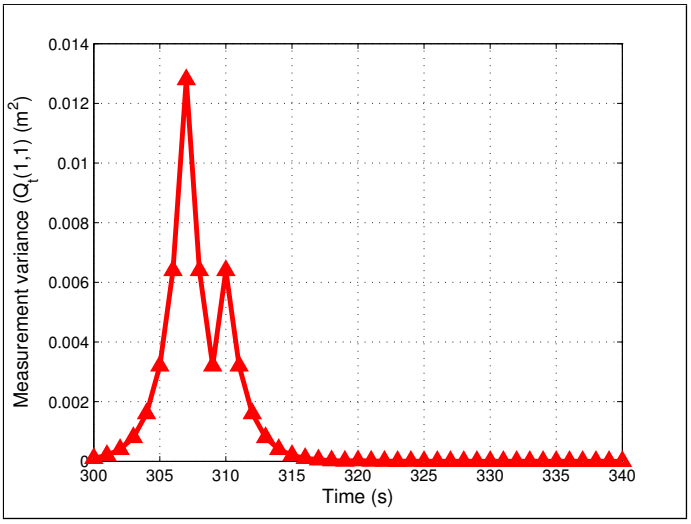

Figure 4.10 Adaptive extended Kalman measurement covariance (Q_t) values during convergence with $Q_0 = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$, $R = 4 \times 10^{-7} \times I_4$, and $M_t = 2$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

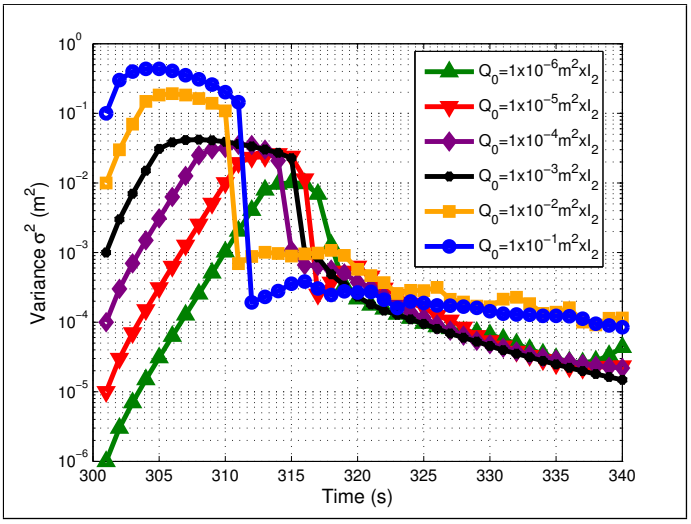

Figure 4.11 Adaptive extended Kalman filter variance (σ^2) from the state covariance matrix (Σ) for a range of starting state model covariance values (Q_0), constant measurement covariance of $R = 4 \times 10^{-7} \times I_4$, and $M_t = 2$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

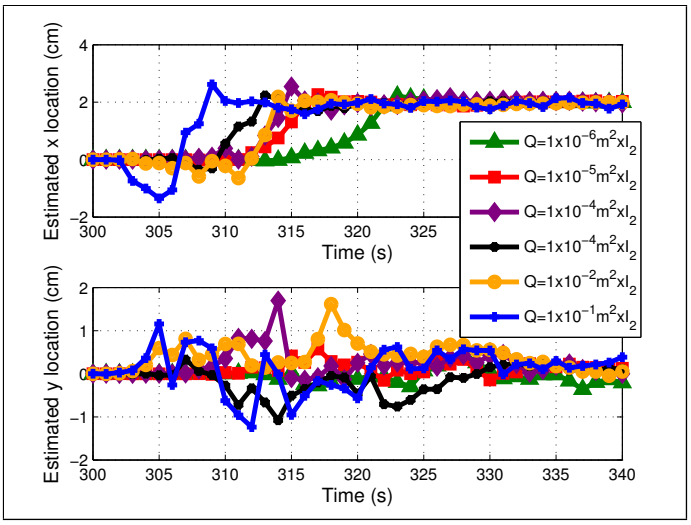

Figure 4.12 Adaptive extended Kalman filter convergence for a range of initial state model covariance values (Q_0), constant measurement covariance of $R = 4 \times 10^{-7} \times I_4$, and a state model covariance gain of $M = 2$. The heating source is located at ($x = 2\,\text{cm}$, $y = 0\,\text{cm}$) with an initial guess of ($x = 0\,\text{cm}$, $y = 0\,\text{cm}$).

extended Kalman filter is able to converge quicker for a significant range of initial state model covariance values Q_0. Figure 4.13 illustrates the sensitivity to the adaptive extended Kalman filter gain M_t.

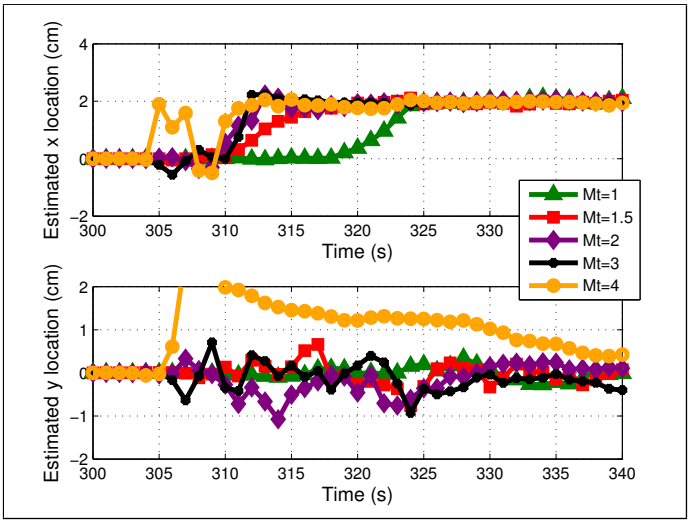

Figure 4.13 Adaptive extended Kalman filter convergence for a range of state model covariance gain values M, an initial state model covariance of $Q_0 = 1 \times 10^{-4}\,\mathrm{m}^2 \times I_2$, and constant measurement covariance of $R = 4 \times 10^{-7} \times I_4$. The heating source is located at ($x = 2\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$) with an initial guess of ($x = 0\,\mathrm{cm}$, $y = 0\,\mathrm{cm}$).

CHAPTER V

CONCLUSIONS

This dissertation and the underlying research and published works contribute to science in five areas: 1) by successfully employing the extended Kalman filter for a transient heat transfer problem, 2) by incorporating a 3D thermal model into the inverse method, 3) by exploring the sensitivity of ultrasound sensors to heating source location, boundary conditions, and thermal conductivity, 4) by addressing uncertainty by modifying the extended Kalman filter to achieve more robust performance, and 5) by detailing an innovative measurement model that produces a closed-form Jacobian.

For locating and characterizing a heating source, the extended Kalman filter produces results similar to least squares, particle filter, and extended information filter. This finding is significant as the ultrasound, extended Kalman filter-based method is pursued further and other heat transfer applications are considered. Kalman filter successes in robotics and other fields lies in its simplicity, computational efficiency, and the ability to include a control input. Whereas this work had no inputs to the state model, the ability to add inputs to the extended Kalman filter is anticipated to be more robust for boundary layer transition region estimation and heating source parameter estimation in general. For a hypersonic vehicle, the control input to the extended Kalman filter could be the pilot's flight control commands (e.g., throttle, attitude controls, etc.) and sensor data (e.g., angle of attack, altitude, etc.). The extended Kalman filter would be appropriate for applications where the user is able to control some aspect of the environment that affects the parameters being estimated (e.g., a hypersonic vehicle). For heat transfer applications without a control input, least squares would be the better choice for the inverse method.

The 3D COMSOL® models developed in this work capture the temperature response of the plate heated with a spot source and the plate heated with the step source with sufficient accuracy to support parameter estimation of the heating source location and other thermal parameters. The method is unique in that a finite element model is used as a forward conduction solution instead of a 1D or 2D closed-form solution. The numerical models are linear but are readily capable of incorporating non-linear effects through material properties and boundary conditions, a feat not shared by the closed-form solutions.

One key finding from the spot source analysis is that sensitivity to heating source location is greater in the direction perpendicular to the ultrasonic pulse propagation path. This finding coupled with knowledge of possible heating source locations can assist designers implementing a system based on this technology. For example, if possible heating source locations are restricted

to locations along the y-axis, the designer would want to employ a set of sensors with propagation paths parallel to the x-axis. The inverse procedure is able to correctly estimate the spot heating source location if the heating source is inside the sensor array. However, the procedure is unable to correctly estimate the spot heating source location if the heating source is outside the sensor array. This finding is interesting and critical to the design of sensor arrays.

The solution is able to track a moving step source with both a sine wave and a sawtooth pattern with a period of 160 s , although overshoot is evident. Expected frequency or speed of transition region movements on a hypersonic vehicle are unknown and these movement patterns represent a starting point for comparison. As the movement frequency increases, the ability to track the moving step location would diminish. Above some frequency, the solution would likely track the median position. Estimation of heat flux magnitude with the one-way ultrasonic pulse sensor array produces satisfactory results for a constant heat flux magnitude but produces significant errors for a variable magnitude. Depending upon the application, this method may not produce satisfactory results. Adding thermocouples collocated with the ultrasonic transducers to measure heat flux would be an effective method for applications that could benefit from increased heat flux accuracy such as hypersonic boundary layer transition region estimation.

The extended Kalman filter variance can be used to drive modifications to the state model covariance in a manner designed to positively affect convergence behavior. The adaptive extended Kalman filter detailed in this work would be beneficial for any application involving transient heating sources since transient performance is improved when the inverse method is capable of quickly converging to the correct location. Non-transient heating source parameter estimation would likely benefit from the adaptive modifications to the extended Kalman filter; however, the advantages are greater when considering transient heating sources.

The ellipse from ultrasonic pulse one-way time of flight measurement model has the unique feature of a closed form Jacobian. The hypothesis brings forth the notion that a closed form Jacobian requires fewer resources while providing convergence similar to the other models that utilize a finite difference Jacobian. Convergence results, however, are unsatisfactory for the ellipse model in this work. The ellipse is an imperfect approximation of the oval shape found in the analysis which may introduce errors in the estimation. The model's efficiency and low wall-times warrant further study for this application and might be more successful when applied to other applications. Additionally, the adaptive extended Kalman filter combined with the ellipse model might produce better results since the adaptive component may overcome errors introduced by the ellipse approximation.

Next steps in developing this solution should include conducting step source experiments identical to the experiments detailed in this work but with ultrasonic transducers, using the ultrasound data in parameter estimation analyses on the step source, applying the adaptive filter

techniques to the step source, and applying the solution to other heating source applications.

APPENDIX

This appendix contains a representative sample of the COMSOL Multiphysics® and MATLAB® code used to perform the analysis in this dissertation.

Code for Spot Heating Source

```
Steps to create numerical model in COMSOL:

3D heat transfer, conduction, transient
Work plane z=0
Rectangle 0.61 x 0.305 centered
Circle 0.003178175 radius centered
Circle 0.06 radius centered
Free mesh parameters - large circle: max element size 5e-3, small circle: max element size 1e-3
Extrude 3 layers, 0.00635 m
Contstants: Tamb, T0, Tinf, Tr all equal to 297.5 [K]
Physics, subdomain: k=14.6 [W/mK], rho=8000 [kg/m^3], Cp=500 [J/kg K], init T(t0)=T0
Physics, boundary:
bottom: h=3.2 [W/m^2 K], Tinf=Tinf, Tamb=Tamb
edges:  h=3.0 [W/m^2 K], Tinf=Tinf, Tamb=Tamb
top:  h=3.2 [W/m^2 K], Tinf=Tinf, Tamb=Tamb
spot: q=0.93e6*(flc2hs(t-300,1)-flc2hs(t-600,1))
Solver: range(0,290,290) range(291,1,800) range(805,5,1400), Time stepping strict, specified times
```

```
%{
Extended Kalman Filter for parameter identification
Mike Myers
17 Jan 2010

%}

close all
clear all
clc

% Initialize random number generator
randn('state',sum(100*clock))

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = 0.0001 * eye(2); % covariance matrix
R = [1e5 0
    0 0.1];

% Define measurement model matrices
Q = 0.25 * eye(5056); % covariance matrix

% Make initial guess for each parameter
% Make initial guess for each parameter
qtext = '931';
htext = '328';

q = str2num(qtext)/1000 * 1e6; % heat flux [W/m^2]
hcoeff = str2num(htext)/100;        % convection coefficient on ...
top and bottom [W/m^2K]
b = [q ; hcoeff ]; % Initial guess of unknown beta vector (beta = b)
B = b;


% Import the experiment data
exp1 = [importdata('../../Nov Experiments/110909_experiment1.csv')];
exp2 = [importdata('../../Nov Experiments/111109_no paint_Air_exp1.csv')];
exp3 = [importdata('../../Nov Experiments/111109_no paint_Air_exp2.csv')];
% Choose which experiment to use
exp = exp2;

% Import COMSOL data
Baseline = [importdata(['q',qtext,'_q100_sig0009_h',htext,....
'_k15_x1to4_y1to4_z00635.txt']); importdata(['q',qtext,'_q100_sig0009_h'...
,htext,'_k15_x1to4_y1to4_z0.txt'])];
dQt = [importdata(['q',qtext,'*1.001_q100_sig0009_h',htext...
,'_k15_x1to4_y1to4_z00635.txt']); importdata(['q',qtext...
,'*1.001_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z0.txt'])];
dHt = [importdata(['q',qtext,'_q100_sig0009_h',htext...
,'*1.001_k15_x1to4_y1to4_z00635.txt']); importdata(['q'..
,qtext,'_q100_sig0009_h',htext,'*1.001_k15_x1to4_y1to4_z0.txt'])];

% Baseline = [Baseline  [importdata('q104_q100_sig0009_h1161_k15_x1to4...
_y1to4_z00635.txt'); ...
importdata('q1_q100_sig0009_h614_k15_x1to4_y1to4_z0.txt')]];
% dQt = [dQt  [importdata('q1*1.001_q100_sig0009_h614_k15_x1to4_y1to4...
_z00635.txt'); ...
importdata('q1*1.001_q100_sig0009_h614_k15_x1to4_y1to4_z0.txt')]];
% dHt = [dHt  [importdata('q1_q100_sig0009_h614*1.001_k15_x1to4_y1to4...
_z00635.txt'); ...
importdata('q1_q100_sig0009_h614*1.001_k15_x1to4_y1to4_z0.txt')]];
%
% Baseline = [Baseline  [importdata('q0939_q100_sig0009_h266_k15_x1to4...
_y1to4_z00635.txt'); importdata('q0939_q100_sig0009_h266_k15_x1to4...
```

```
_y1to4_z0.txt')]];
% Qt = [Qt  [importdata('q0939*1.001_q100_sig0009_h266_k15_x1to4_y1to4...
_z00635.txt'); importdata('q0939*1.001_q100_sig0009_h266_k15_x1to4...
_y1to4_z0.txt')]];
% Ht = [Ht  [importdata('q0939_q100_sig0009_h266*1.001_k15_x1to4_y1to4...
_z00635.txt'); importdata('q0939_q100_sig0009_h266*1.001_k15_x1to4...
_y1to4_z0.txt')]];

% Collect the experiment data for the same timesteps in COMSOL
j=1;
experiment = [];
for i=1:length(exp)
    if exp(i,1) == Baseline(j,1)
        experiment = [experiment;exp(i,:)];
        j=j+1;
    end
end
experiment = [experiment(:,2);experiment(:,3);experiment(:,4);...
experiment(:,5);experiment(:,6);experiment(:,7);experiment(:,8);...
experiment(:,9)];

% Initialize EKF parameters
mu0 = [q
    hcoeff];
Sigma0 = R;%0;
Mu = mu0;
Mubar = [];
mutminus1 = mu0;
Sigmatminus1 = Sigma0;
h=[];
Ht=[];
Kt=[];
Zt=[];

for i=1:size(Baseline,2)/2

    baseline = Baseline(:,2*i-1:2*i);
    qt = dQt(:,2*i-1:2*i);
    ht = dHt(:,2*i-1:2*i);
    fprintf(['Run ', num2str(i), ':                q=%2.2e   h= %2.2f  '], ...
     mutminus1(1), mutminus1(2));

    % experiment has duplicate set of sensors at (.01,.01) ...
     so reorganize COMSOL data
    n = length(qt)/8;
    baseline = [baseline(1:n,:) ; baseline(1:n,:) ; baseline(n+1:2*n,:) ; ...
     baseline(2*n+1:3*n,:) ; baseline(4*n+1:5*n,:) ; ...
     baseline(4*n+1:5*n,:) ; baseline(5*n+1:6*n,:) ; ...
     baseline(6*n+1:7*n,:) ];
    qt = [qt(1:n,:) ; qt(1:n,:) ; qt(n+1:2*n,:) ; qt(2*n+1:3*n,:) ; ...
     qt(4*n+1:5*n,:) ; qt(4*n+1:5*n,:) ; qt(5*n+1:6*n,:) ; ...
     qt(6*n+1:7*n,:) ];
    ht = [ht(1:n,:) ; ht(1:n,:) ; ht(n+1:2*n,:) ; ht(2*n+1:3*n,:) ; ...
     ht(4*n+1:5*n,:) ; ht(4*n+1:5*n,:) ; ht(5*n+1:6*n,:) ; ...
     ht(6*n+1:7*n,:) ];

    % Compute and display MSE and MAE for the baseline
    mse = sum( (experiment(:) - baseline(:,2)).^2 ) / length(experiment);
    mae = sum( abs( experiment(:) - baseline(:,2) ) ) / length(experiment);
    fprintf([' mse=%2.2f   mae= %2.2f \n'], mse, mae);

    tic;
    % Kalman filter prediction
    mubart = g * mutminus1;
    Sigmabart = G * Sigmatminus1 * G' + R;

    % Compute measurement function (h) and measurement Jacobian (H)
    h = baseline(:,2);
```

```
    dTdq = ( qt(:,2) - baseline(:,2) ) / (q * 0.001) ;
    dTdh = ( ht(:,2) - baseline(:,2) ) / (hcoeff * 0.001) ;
    Ht = [ dTdq , dTdh ];

    % Ht = [ (( qt(:,2) - baseline(:,2) ) / (q * 0.001))   ...
     (( ht(:,2) - baseline(:,2) ) / (hcoeff * 0.001))];

    % Compute Kalman gain
    Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

    % Get a measurement update using experiment
    Zt = experiment;

    % Kalman filter correction
    mut = mubart + Kt * (Zt - h);
    Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

    Mu = [ Mu mut];
    Mubar = [Mubar mubart];
    Sigmatminus1 = Sigmat;
    mutminus1 = mut;

    wall_time = toc;
    % Compute the new parameters (b vector)
    % b = bn .* b;
    fprintf(['Kalman update:    q=%2.2e   h= %2.2f      ...
     wall_time=%2.3e sec\n'], mut(1), mut(2), wall_time)

    % Compute and plot the confidence intervals
    t = tinv(0.95,length(experiment)-1);
    s = diag(inv(Ht'*Ht) * mse);
    interval = t*sqrt( s );
    C = [mut - interval, mut + interval];
    fprintf(['Interval:               q=%2.2f     h= %2.2f \n'], ...
     interval(1), interval(2));
    fprintf(['Confidence lb:      q=%2.2e   h= %2.2f \n'], C(1,1), C(2,1));
    fprintf(['Confidence ub:     q=%2.2e   h= %2.2f \n\n'], C(1,2), C(2,2));

    Cn(1,:) = C(1,:)./Mu(1,i);
    Cn(2,:) = C(2,:)./Mu(2,i);

%     figure
%     errorbar(1, (Cn(1,1)+Cn(1,2))/2, interval(1)/Mu(1,i), 'Linewidth',2)
%     hold on
%     errorbar(2, (Cn(2,1)+Cn(2,2))/2, interval(2)/Mu(2,i), 'Linewidth',2)
%     grid on
%     xlabel('\beta Parameter (q, h)','FontSize',12)
%     ylabel('\beta_{normalized}','FontSize',12)
%     title(['Confidence Intervals for \beta Parameters'],'FontSize',12)
%
%     figure
%     plot(baseline(1:n,1), baseline(1:n,2),'r','Linewidth',2)
%     hold on;
%     plot(baseline(1:n,1), experiment(1:n),'b','Linewidth',2)
%     plot(baseline(n+1:2*n,1), baseline(n+1:2*n,2),'r','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), baseline(2*n+1:3*n,2),'r','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), baseline(3*n+1:4*n,2),'r','Linewidth',2)
%     plot(baseline(n+1:2*n,1), experiment(n+1:2*n),'b','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), experiment(2*n+1:3*n),'b','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), experiment(3*n+1:4*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Temperature Response on Non-Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')
%
%     figure
%     plot(baseline(4*n+1:5*n,1), baseline(4*n+1:5*n,2),'r','Linewidth',2)
```

```
%     hold on;
%     plot(baseline(4*n+1:5*n,1), experiment(4*n+1:5*n),'b','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), baseline(5*n+1:6*n,2),'r','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), baseline(6*n+1:7*n,2),'r','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), baseline(7*n+1:8*n,2),'r','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), experiment(5*n+1:6*n),'b','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), experiment(6*n+1:7*n),'b','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), experiment(7*n+1:8*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Temperature Response on Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')

end

fprintf(['Final parameters: q=%2.2e   h= %2.2f \n'], Mu(1,i+1), Mu(2,i+1));
```

```
%{
Extended Information Filter for parameter identification
Mike Myers
13 Jan 2010

%}

clear all
close all
clc

% Initialize random number generator
randn('state',sum(100*clock))

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = [1e5 0 % state covariance
    0 0.1];

% Define measurement model matrices
Q = 0.25 * eye(5056); % covariance matrix
Qinv = 1/0.25 * eye(5056); % covariance matrix inverse

% Make initial guess for each parameter
qtext = '175';
htext = '50';

q = str2num(qtext)/100 * 1e6; % heat flux [W/m^2]
hcoeff = str2num(htext)/10;        % convection coefficient on top and bottom [W/m^2K]

LB_q = [];
UB_q = [];
LB_h = [];
UB_h = [];

% Import the experiment data
exp1 = [importdata('../../Nov Experiments/110909_experiment1.csv')];
exp2 = [importdata('../../Nov Experiments/111109_no paint_Air_exp1.csv')];
exp3 = [importdata('../../Nov Experiments/111109_no paint_Air_exp2.csv')];
% Choose which experiment to use
exp = exp2;

% Import COMSOL data
Baseline = [importdata(['q',qtext,'_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z00635.txt']);...
 importdata(['q',qtext,'_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z0.txt'])];
dQt = [importdata(['q',qtext,'*1.001_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z00635.txt']); ...
importdata(['q',qtext,'*1.001_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z0.txt'])];
dHt = [importdata(['q',qtext,'_q100_sig0009_h',htext,'*1.001_k15_x1to4_y1to4_z00635.txt']); ...
importdata(['q',qtext,'_q100_sig0009_h',htext,'*1.001_k15_x1to4_y1to4_z0.txt'])];

% Baseline = [Baseline  [importdata('q0940_q100_sig0009_h272_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0940_q100_sig0009_h272_k15_x1to4_y1to4_z0.txt')]];
% dQt = [dQt  [importdata('q0940*1.001_q100_sig0009_h272_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0940*1.001_q100_sig0009_h272_k15_x1to4_y1to4_z0.txt')]];
% dHt = [dHt  [importdata('q0940_q100_sig0009_h272*1.001_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0940_q100_sig0009_h272*1.001_k15_x1to4_y1to4_z0.txt')]];
%
% Baseline = [Baseline  [importdata('q0940_q100_sig0009_h266_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0940_q100_sig0009_h266_k15_x1to4_y1to4_z0.txt')]];
% dQt = [dQt  [importdata('q0940*1.001_q100_sig0009_h266_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0940*1.001_q100_sig0009_h266_k15_x1to4_y1to4_z0.txt')]];
% dHt = [dHt  [importdata('q0940_q100_sig0009_h266*1.001_k15_x1to4_y1to4_z00635.txt');...
%  importdata('q0940_q100_sig0009_h266*1.001_k15_x1to4_y1to4_z0.txt')]];

% Collect the experiment data for the same timesteps in COMSOL
j=1;
```

```
experiment = [];
for i=1:length(exp)
    if exp(i,1) == Baseline(j,1)
        experiment = [experiment;exp(i,:)];
        j=j+1;
    end
end
experiment = [experiment(:,2);experiment(:,3);experiment(:,4);experiment(:,5);experiment(:,6);...
experiment(:,7);experiment(:,8);experiment(:,9)];

% Initialize EIF parameters
mu0 = [q
    hcoeff];
Sigma0 = R;%1000*eye(2);
Mu = mu0;
Mubar = [];
mutminus1 = mu0;
Sigmatminus1 = Sigma0;
h=[];
Ht=[];
Kt=[];
Zt=[];

for i=1:size(Baseline,2)/2

    baseline = Baseline(:,2*i-1:2*i);
    qt = dQt(:,2*i-1:2*i);
    ht = dHt(:,2*i-1:2*i);
    fprintf(['Run ', num2str(i), ':                q=%2.3e   h= %2.2f  '], mutminus1(1), mutminus1(2));

    % experiment has duplicate set of sensors at (.01,.01) so reorganize COMSOL data
    n = length(qt)/8;
    baseline = [baseline(1:n,:) ; baseline(1:n,:) ; baseline(n+1:2*n,:) ; ...
     baseline(2*n+1:3*n,:) ; baseline(4*n+1:5*n,:) ; baseline(4*n+1:5*n,:) ; ...
baseline(5*n+1:6*n,:) ; baseline(6*n+1:7*n,:) ];
    qt = [qt(1:n,:) ; qt(1:n,:) ; qt(n+1:2*n,:) ; qt(2*n+1:3*n,:) ; qt(4*n+1:5*n,:) ; qt(4*n+1:5*n,:) ; ...
     qt(5*n+1:6*n,:) ; qt(6*n+1:7*n,:) ];
    ht = [ht(1:n,:) ; ht(1:n,:) ; ht(n+1:2*n,:) ; ht(2*n+1:3*n,:) ; ht(4*n+1:5*n,:) ; ht(4*n+1:5*n,:) ; ...
     ht(5*n+1:6*n,:) ; ht(6*n+1:7*n,:) ];

    % Compute and display MSE and MAE for the baseline
    mse = sum( (experiment(:) - baseline(:,2)).^2 ) / length(experiment);
    mae = sum( abs( experiment(:) - baseline(:,2) ) ) / length(experiment);
    fprintf([' mse=%2.2f   mae= %2.2f \n'], mse, mae);

    tic
    Omegatminus1 = inv(Sigmatminus1);
    xitminus1 = Sigmatminus1 \ mutminus1;

    % Extended Information Filter
    mutminus1 = Omegatminus1 \ xitminus1;
    Omegabart = inv( G / Omegatminus1 * G' + R );
    xibart = Omegabart * (g * mutminus1);
    mubart = ( g * mutminus1);

    % Compute measurement function (h) and measurement Jacobian (H)
    h = baseline(:,2);
    dTdq = ( qt(:,2) - baseline(:,2) ) / (q * 0.001) ;
    dTdh = ( ht(:,2) - baseline(:,2) ) / (hcoeff * 0.001) ;
    Ht = [ dTdq , dTdh ];

    % Get a measurement update using experiment
    Zt = experiment;

    % EIF correction
    Omegat = Omegabart + Ht' * Qinv * Ht;
    xit = xibart + Ht' * Qinv * (Zt - h + Ht * mubart);
```

```
    % Recover state vector and covariance matrix
    mut = Omegat \ xit;
    Sigmat = inv(Omegat);

    Mu = [ Mu mut];
    Mubar = [Mubar mubart];
    Sigmatminus1 = Sigmat;
    mutminus1 = mut;
    wall_time = toc;

    fprintf(['EIF update:          q=%2.3e    h= %2.2f      ...
     wall_time=%2.3e sec\n'], mut(1), mut(2), wall_time)
    fprintf(['delta (abs):          q=%2.3e    h= %2.2f \n'], ...
     abs(Mu(1,i+1)-Mu(1,i)), abs(Mu(2,i+1)-Mu(2,i)));

    % Compute and plot the confidence intervals
    t = tinv(0.95,length(experiment)-1);
    s = diag(inv(Ht'*Ht) * mse);
    interval = t*sqrt( s );
    C = [mut - interval, mut + interval];
    LB_q = [ LB_q C(1,1) ];
    UB_q = [ UB_q C(1,2) ];
    LB_h = [ LB_h C(2,1) ];
    UB_h = [ UB_h C(2,2) ];
    fprintf(['Interval:              q=%2.3f     h= %2.2f \n'], interval(1), interval(2));
    fprintf(['Confidence lb:      q=%2.3e   h= %2.2f \n'], C(1,1), C(2,1));
    fprintf(['Confidence ub:     q=%2.3e   h= %2.2f \n\n'], C(1,2), C(2,2));

    Cn(1,:) = C(1,:)./Mu(1,i);
    Cn(2,:) = C(2,:)./Mu(2,i);

%     figure
%     errorbar(1, (Cn(1,1)+Cn(1,2))/2, interval(1)/Mu(1,i), 'Linewidth',2)
%     hold on
%     errorbar(2, (Cn(2,1)+Cn(2,2))/2, interval(2)/Mu(2,i), 'Linewidth',2)
%     grid on
%     xlabel('\beta Parameter (q, h)','FontSize',12)
%     ylabel('\beta_{normalized}','FontSize',12)
%     title(['Confidence Intervals for \beta Parameters'],'FontSize',12)
%
%     figure
%     plot(baseline(1:n,1), baseline(1:n,2),'r','Linewidth',2)
%     hold on;
%     plot(baseline(1:n,1), experiment(1:n),'b','Linewidth',2)
%     plot(baseline(n+1:2*n,1), baseline(n+1:2*n,2),'r','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), baseline(2*n+1:3*n,2),'r','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), baseline(3*n+1:4*n,2),'r','Linewidth',2)
%     plot(baseline(n+1:2*n,1), experiment(n+1:2*n),'b','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), experiment(2*n+1:3*n),'b','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), experiment(3*n+1:4*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Temperature Response on Non-Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')
%
%     figure
%     plot(baseline(4*n+1:5*n,1), baseline(4*n+1:5*n,2),'r','Linewidth',2)
%     hold on;
%     plot(baseline(4*n+1:5*n,1), experiment(4*n+1:5*n),'b','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), baseline(5*n+1:6*n,2),'r','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), baseline(6*n+1:7*n,2),'r','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), baseline(7*n+1:8*n,2),'r','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), experiment(5*n+1:6*n),'b','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), experiment(6*n+1:7*n),'b','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), experiment(7*n+1:8*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
```

```
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Temperature Response on Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')
%
end

fprintf(['Final parameters: q=%2.3e   h= %2.2f \n'], Mu(1,i+1), Mu(2,i+1));

% figure
% plot(Mu(1,:)/Mu(1,i+1),'-bs','Linewidth',2)
% hold on
% errorbar([2:4], Mu(1,2:4)/Mu(1,4), (UB_q-Mu(1,2:4))/Mu(1,4),'Linewidth',2 )
% set(gca,'XTick',1:1:i+1)
% set(gca,'XTickLabel',{'Initial Guess','1','2','3'})
% xlabel('Iteration','FontSize',12)
% ylabel('Parameter (normalized by q/q_{final})','FontSize',12)
% grid on
%
% figure
% plot(Mu(2,:)/Mu(2,i+1),'-bs','Linewidth',2)
% hold on
% errorbar([2:4], Mu(2,2:4)/Mu(2,4), (UB_h-Mu(2,2:4))/Mu(2,4),'Linewidth',2 )
% set(gca,'XTick',1:1:i+1)
% set(gca,'XTickLabel',{'Initial Guess','1','2','3'})
% xlabel('Iteration','FontSize',12)
% ylabel('Parameter (normalized by h/h_{final})','FontSize',12)
% grid on
```

```
%{
Parameter identification to find best match between the COMSOL
model and one November flat plate experiment. Least squares
method used.

Mike Myers
22 Dec 2009
Updated 17 Jan 2010

%}

close all;
clear all;
clc;

% Make initial guess for each parameter
qtext = '935';
htext = '332';

q = str2num(qtext)/1000 * 1e6; % heat flux [W/m^2]
h = str2num(htext)/100;        % convection coefficient on top and bottom [W/m^2K]

b = [q ; h ]; % Initial guess of unknown beta vector (beta = b)
B = b;
LB_q = [];
UB_q = [];
LB_h = [];
UB_h = [];

% Import the experiment data
exp1 = [importdata('../../Nov Experiments/110909_experiment1.csv')];
exp2 = [importdata('../../Nov Experiments/111109_no paint_Air_exp1.csv')];
exp3 = [importdata('../../Nov Experiments/111109_no paint_Air_exp2.csv')];
% Choose which experiment to use
exp = exp2;

% Import COMSOL data
Baseline = [importdata(['q',qtext,'_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z00635.txt']); ...
importdata(['q',qtext,'_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z0.txt'])];
Qt = [importdata(['q',qtext,'*1.001_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z00635.txt']);...
 importdata(['q',qtext,'*1.001_q100_sig0009_h',htext,'_k15_x1to4_y1to4_z0.txt'])];
Ht = [importdata(['q',qtext,'_q100_sig0009_h',htext,'*1.001_k15_x1to4_y1to4_z00635.txt']);...
 importdata(['q',qtext,'_q100_sig0009_h',htext,'*1.001_k15_x1to4_y1to4_z0.txt'])];

% Baseline = [Baseline  [importdata('q0937_q100_sig0009_h254_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0937_q100_sig0009_h254_k15_x1to4_y1to4_z0.txt')]];
% Qt = [Qt  [importdata('q0937*1.001_q100_sig0009_h254_k15_x1to4_y1to4_z00635.txt'); ...
% importdata('q0937*1.001_q100_sig0009_h254_k15_x1to4_y1to4_z0.txt')]];
% Ht = [Ht  [importdata('q0937_q100_sig0009_h254*1.001_k15_x1to4_y1to4_z00635.txt');...
%  importdata('q0937_q100_sig0009_h254*1.001_k15_x1to4_y1to4_z0.txt')]];
%
% Baseline = [Baseline  [importdata('q0939_q100_sig0009_h266_k15_x1to4_y1to4_z00635.txt');...
%  importdata('q0939_q100_sig0009_h266_k15_x1to4_y1to4_z0.txt')]];
% Qt = [Qt  [importdata('q0939*1.001_q100_sig0009_h266_k15_x1to4_y1to4_z00635.txt');...
%  importdata('q0939*1.001_q100_sig0009_h266_k15_x1to4_y1to4_z0.txt')]];
% Ht = [Ht  [importdata('q0939_q100_sig0009_h266*1.001_k15_x1to4_y1to4_z00635.txt');...
%  importdata('q0939_q100_sig0009_h266*1.001_k15_x1to4_y1to4_z0.txt')]];

% Collect the experiment data for the same timesteps in COMSOL
j=1;
experiment = [];
for i=1:length(exp)
    if exp(i,1) == Baseline(j,1)
        experiment = [experiment;exp(i,:)];
        j=j+1;
    end
end
```

```
experiment = [experiment(:,2);experiment(:,3);experiment(:,4);experiment(:,5);...
experiment(:,6);experiment(:,7);experiment(:,8);experiment(:,9)];

for i=1:size(Baseline,2)/2
    bn = b./b; % normalized b vector
    fprintf(['Run ', num2str(i), ':                q=%2.2e   h= %2.2f  '], b(1), b(2));

    baseline = Baseline(:,2*i-1:2*i);
    qt = Qt(:,2*i-1:2*i);
    ht = Ht(:,2*i-1:2*i);

    % experiment has duplicate set of sensors at (.01,.01) so reorganize COMSOL data
    n = length(qt)/8;
    baseline = [baseline(1:n,:) ; baseline(1:n,:) ; baseline(n+1:2*n,:) ; baseline(2*n+1:3*n,:) ; ...
     baseline(4*n+1:5*n,:) ; baseline(4*n+1:5*n,:) ; baseline(5*n+1:6*n,:) ; baseline(6*n+1:7*n,:) ];
    qt = [qt(1:n,:) ; qt(1:n,:) ; qt(n+1:2*n,:) ; qt(2*n+1:3*n,:) ; qt(4*n+1:5*n,:) ; qt(4*n+1:5*n,:) ; ...
     qt(5*n+1:6*n,:) ; qt(6*n+1:7*n,:) ];
    ht = [ht(1:n,:) ; ht(1:n,:) ; ht(n+1:2*n,:) ; ht(2*n+1:3*n,:) ; ht(4*n+1:5*n,:) ; ht(4*n+1:5*n,:) ; ...
     ht(5*n+1:6*n,:) ; ht(6*n+1:7*n,:) ];

    % Compute and display MSE and MAE for the initial guess
    mse = sum( (experiment(:) - baseline(:,2)).^2 ) / length(experiment);
    mae = sum( abs( experiment(:) - baseline(:,2) ) ) / length(experiment);
    fprintf([' mse=%2.2f   mae= %2.2f \n'], mse, mae);

    % Plot the baseline and experiment data
%     figure
%     plot(baseline(1:n,1), baseline(1:n,2),'r','Linewidth',2)
%     hold on;
%     plot(baseline(1:n,1), experiment(1:n),'b','Linewidth',2)
%     plot(baseline(n+1:2*n,1), baseline(n+1:2*n,2),'r','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), baseline(2*n+1:3*n,2),'r','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), baseline(3*n+1:4*n,2),'r','Linewidth',2)
%     plot(baseline(n+1:2*n,1), experiment(n+1:2*n),'b','Linewidth',2)
%     plot(baseline(2*n+1:3*n,1), experiment(2*n+1:3*n),'b','Linewidth',2)
%     plot(baseline(3*n+1:4*n,1), experiment(3*n+1:4*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Run ', num2str(i),' Temperature Response on Non-Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')
%     figure
%     plot(baseline(4*n+1:5*n,1), baseline(4*n+1:5*n,2),'r','Linewidth',2)
%     hold on;
%     plot(baseline(4*n+1:5*n,1), experiment(4*n+1:5*n),'b','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), baseline(5*n+1:6*n,2),'r','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), baseline(6*n+1:7*n,2),'r','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), baseline(7*n+1:8*n,2),'r','Linewidth',2)
%     plot(baseline(5*n+1:6*n,1), experiment(5*n+1:6*n),'b','Linewidth',2)
%     plot(baseline(6*n+1:7*n,1), experiment(6*n+1:7*n),'b','Linewidth',2)
%     plot(baseline(7*n+1:8*n,1), experiment(7*n+1:8*n),'b','Linewidth',2)
%     grid on
%     xlabel('Time (s)','FontSize',12)
%     ylabel('Temperature Change, T-T0 (K)','FontSize',12)
%     title(['Run ', num2str(i),' Temperature Response on Heated Side'],'FontSize',12)
%     legend('Model', 'Experiment','Location', 'Northeast')

    tic; % initialize timer for wall time
    % Sensitivity matrix X (normalized so units are K)
    dTdq = ( qt(:,2) - baseline(:,2) ) / (q * 0.001) * q;
    dTdh = ( ht(:,2) - baseline(:,2) ) / (h * 0.001) * h;
    X = [ dTdq , dTdh ];

    % Compute new b vector (normalized)
    deltabn = inv(X'*X)*X'*(experiment - baseline(:,2)); % could also use X\(experiment - baseline(:,2))
    bn = bn + deltabn;
    fprintf(['delta b (abs):       q=%2.2f           h= %2.2f \n'], abs(deltabn(1)), abs(deltabn(2)));
```

```
    % Compute the new parameters (b vector)
    b = bn .* b;
    B = [ B b ];
    wall_time = toc;
    fprintf(['Gaussian update: q=%2.2e   h= %2.2f     wall_time=%2.3e sec\n'], b(1), b(2), wall_time);

    % Compute and plot the confidence intervals
    t = tinv(0.95,length(experiment)-1);
    s = diag(inv(X'*X) * mse);
    interval = t*sqrt( s );
    Cn = [bn - interval, bn + interval];
    LB_q = [ LB_q Cn(1,1)*B(1,i) ];
    UB_q = [ UB_q Cn(1,2)*B(1,i) ];
    LB_h = [ LB_h Cn(2,1)*B(2,i) ];
    UB_h = [ UB_h Cn(2,2)*B(2,i) ];

    fprintf(['Interval (norm):     q=%2.2f           h= %2.2f \n'], interval(1), interval(2));
    fprintf(['Confidence lb:      q=%2.2e   h= %2.2f \n'], Cn(1,1)*B(1,i), Cn(2,1)*B(2,i));
    fprintf(['Confidence ub:     q=%2.2e   h= %2.2f \n\n'], Cn(1,2)*B(1,i), Cn(2,2)*B(2,i));
%     figure
%     errorbar(1, bn(1), interval(1), 'Linewidth',2)
%     hold on
%     errorbar(2, bn(2), interval(2), 'Linewidth',2)
%     grid on
%     xlabel('Parameter (q, h)','FontSize',12)
%     ylabel('b vector (normalized)','FontSize',12)
%     title(['Run ', num2str(i),' Normalized Parameter Confidence Intervals'],'FontSize',12)

end

fprintf(['Final parameters: q=%2.2e   h= %2.2f \n'], B(1,i+1), B(2,i+1));

% figure
% plot(B(1,:)/B(1,i+1),'-bs','Linewidth',2)
% hold on
% errorbar([2:4], B(1,2:4)/B(1,4), (UB_q-B(1,2:4))/B(1,4),'Linewidth',2 )
% set(gca,'XTick',1:1:i+1)
% set(gca,'XTickLabel',{'Initial Guess','1','2','3'})
% xlabel('Iteration','FontSize',12)
% ylabel('Parameter (normalized by q/q_{final})','FontSize',12)
% grid on
%
% figure
% plot(B(2,:)/B(2,i+1),'-bs','Linewidth',2)
% hold on
% errorbar([2:4], B(2,2:4)/B(2,4), (UB_h-B(2,2:4))/B(2,4),'Linewidth',2 )
% set(gca,'XTick',1:1:i+1)
% set(gca,'XTickLabel',{'Initial Guess','1','2','3'})
% xlabel('Iteration','FontSize',12)
% ylabel('Parameter (normalized by h/h_{final})','FontSize',12)
% grid on
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Temperature measurement model

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 0.045 % K
measurement_variance = 2.225e-4; % K^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define sensor locations (cm)
n_sensors = 4;
x = [ 0.04 , -0.04 , -0.04 ,  0.04 ];
y = [ 0.04 ,  0.04 , -0.04 , -0.04 ];
z = [ 0.00635      ,  0.00635      ,   0.00635      ,   0.00635      ];

% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];

dx = 0.0001;
dy = 0.0001;

% Define time steps to locate source (s)
t=[300:1:300];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define radius values for indirect measurements
rmax = 0.1;
r = linspace(0,rmax,200);

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % state covariance matrix

% Define measurement model matrices
% Q = 0.0001 * eye(4); % covariance matrix
Q = measurement_variance * eye(4); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.0
        0.0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
```

```
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Hn=[];
    Kn=[];
    Zn=[];
    walltime = [];

    fprintf('  n    mut_x       mut_y  \n');

    for i = 1:length(t)
        tic
        for n=1:1
            % Kalman filter prediction
            mubart = g * mutminus1;
            Sigmabart = G * Sigmatminus1 * G' + R;

            h = zeros(n_sensors,1);
            Ht = zeros(n_sensors,2);
%             for k = 1:n_sensors
                % Compute measurement function (h) and measurement Jacobian (H)
                h = postinterp(fem, 'T',[x-mubart(1) ; y-mubart(2) ; z ], 'T', t(i))';
                Ht(:,1) = -(postinterp(fem, 'T',[x+dx-mubart(1) ; y-mubart(2) ; z ], 'T', t(i))' - h) / dx;
                Ht(:,2) = -(postinterp(fem, 'T',[x-mubart(1) ; y+dy-mubart(2) ; z ], 'T', t(i))' - h) / dy;

                % Get a measurement update using the simulated source
                Zt =  postinterp(fem, 'T',[x-xs(j) ; y-ys(j) ; z ], 'T', t(i))';
%             end

            Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

            % Compute Kalman gain
            Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

            % Kalman filter correction
            mut = mubart + Kt * (Zt - h);
            Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

            % Limit estimated source location to points on the plate
            if abs(mut(1)) > xmax
                mut(1) = xmax*sign(mut(1));
            end
            if abs(mut(2)) > ymax
                mut(2) = ymax*sign(mut(2));
            end

            Mu = [ Mu mut];
            Mubar = [Mubar mubart];
            Sigmatminus1 = Sigmat;
            mutminus1 = mut;

            %         fprintf('%3.0f   %3.3f    %3.3f \n',n, mut(1), mut(2))
        end
        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;
    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)
```

```
    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

% save 'TC_radius_00.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Radius from temperature measurement model

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 0.045 % K
measurement_variance = 5.06e-8; % m^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define sensor locations (cm)
n_sensors = 4;
x = [ 0.04 , -0.04 , -0.04 ,  0.04 ];
y = [ 0.04 ,  0.04 , -0.04 , -0.04 ];
z = [ 0.00635      ,  0.00635      ,   0.00635      ,   0.00635      ];

% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];


% Define time steps to locate source (s)
t=[300:1:360];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define radius values for indirect measurements
rmax = 0.1;
r = linspace(0,rmax,200);

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % state covariance matrix

% Define measurement model matrices
% Q = 0.0001 * eye(4); % covariance matrix
Q = measurement_variance * eye(4); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.08
        0.08];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
```

```
h=[];
Hn=[];
Kn=[];
Zn=[];
walltime = [];

fprintf('  n    mut_x       mut_y  \n');

for i = 1:length(t)
    tic
    for n=1:1
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        % Compute measurement function (h) and measurement Jacobian (H)
        h = [sqrt( (x(1) - mubart(1))^2 + (y(1) - mubart(2))^2 )
            sqrt( (x(2) - mubart(1))^2 + (y(2) - mubart(2))^2 )
            sqrt( (x(3) - mubart(1))^2 + (y(3) - mubart(2))^2 )
            sqrt( (x(4) - mubart(1))^2 + (y(4) - mubart(2))^2 ) ];
        Hn = [ ( mubart(1) - x(1) ) / h(1)     ( mubart(2) - y(1) ) / h(1)
            ( mubart(1) - x(2) ) / h(2)     ( mubart(2) - y(2) ) / h(2)
            ( mubart(1) - x(3) ) / h(3)     ( mubart(2) - y(3) ) / h(3)
            ( mubart(1) - x(4) ) / h(4)     ( mubart(2) - y(4) ) / h(4) ];

        % Compute Kalman gain
        Kn = Sigmabart * Hn' * inv( Hn * Sigmabart * Hn' + Q );

        % Get a measurement update using the simulated source
        Zn = postinterp(fem, 'T-Tinf',[x-xs(j) ; y-ys(j) ; z ], 'T', t(i))';
        Zn = Zn + sensor_noise .* randn(n_sensors,1); % add noise to temperature measurements

        % Use COMSOL to lookup radius using measured temperature
        T = postinterp(fem, 'T-Tinf',[ r ; zeros(1, length(r)) ; zeros(1, length(r)) ], 'T', t(i))';
        Zn = interp1( T, r, Zn, 'spline');

        % Limit radius to rmax
        if max(Zn) > rmax
            for Zi = 1:length(Zn)
                if Zn(Zi) > rmax
                    Zn(Zi) = rmax;
                end
            end
        end

        % Kalman filter correction
        mut = mubart + Kn * (Zn - h);
        Sigmat = (eye(2) - Kn * Hn) * Sigmabart;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
        end

        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        %         fprintf('%3.0f   %3.3f    %3.3f \n',n, mut(1), mut(2))
    end
    if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
        converged = [];
    else
        converged = 'unconverged';
```

```
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;
    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

save 'TC_radius_88.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Ultrasonic pulse-echo time of flight measurement model

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 2.3e-10; % sec
measurement_variance = 5.88e-21; % sec^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)

L = 2*0.00635; % distance ultrasonic pulse travels (m)

dx = 0.0001;
dy = 0.0001;

% Define sensor locations (m)
n = 100; % number of points in plate thickness to retrieve temperature
%(for computing average temp in plate)
d = 0.04; % sensor location distance from origin
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, -d, n);
z(1,:) = linspace(0.00635, 0, n);

x(2,:) = linspace(-d,-d,n);
y(2,:) = linspace(d,d,n);
z(2,:) = linspace(0.00635, 0, n);

x(3,:) = linspace(d,d,n);
y(3,:) = linspace(d,d,n);
z(3,:) = linspace(0.00635, 0, n);

x(4,:) = linspace(d,d,n);
y(4,:) = linspace(-d,-d,n);
z(4,:) = linspace(0.00635, 0, n);

% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);
```

```
% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % covariance matrix

% Define measurement model matrices
% Q = 1.78e-20 * eye(n_sensors); % covariance matrix
Q = measurement_variance * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.08
        0.08];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_x      mut_y  \n');

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        h = [];
        Ht = [];
        for k = 1:n_sensors
            h(k,1) = L/v0 * (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0));
            Ht(k,1) = -(L/v0 * (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ; ...
             y(k,:)-mubart(2) ; ...
             z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dx;
            Ht(k,2) = -(L/v0 * (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)+dy-mubart(2) ; ...
             z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dy;

            % Get a measurement update using the simulated source
            Zt(k,1) =  L/v0 * (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ; y(k,:)-ys(j) ; ...
             z(k,:) ], 'T', t(i)))-T0));
        end

        Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

        % Kalman filter correction
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
        end
```

```
        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

save 'PulseEchoTOF_88.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Pulse-echo TOF radius measurement model

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 2.3e-10; % sec
measurement_variance = 1.76e-5; % m^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)

% Define sensor locations (cm)
zn = 100; % number of points in plate thickness to retrieve temperature
% (for computing average temp in plate)
d = 0.04; % sensor location distance from origin
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, zn);
y(1,:) = linspace(-d, -d, zn);
z(1,:) = linspace(0.00635, 0, zn);

x(2,:) = linspace(-d,-d, zn);
y(2,:) = linspace(d, d, zn);
z(2,:) = linspace(0.00635, 0, zn);

x(3,:) = linspace(d, d, zn);
y(3,:) = linspace(d, d, zn);
z(3,:) = linspace(0.00635, 0, zn);

x(4,:) = linspace(d, d, zn);
y(4,:) = linspace(-d, -d, zn);
z(4,:) = linspace(0.00635, 0, zn);


% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define radius values for indirect measurements
rmax = 0.1;
r = linspace(0,rmax,zn);
rx = meshgrid(r);
```

```
rx = rx(:)';
ry = linspace(-d, d, zn);
ry = meshgrid(ry)';
ry = ry(:)';
rz = meshgrid(z(1,:));
rz = rz(:)';

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = 0.0001 * eye(2); % state covariance matrix

% Define measurement model matrices
Q = 0.0001 * eye(4); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.08
        0.08];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Hn=[];
    Kn=[];
    Zn=[];
    walltime = [];

    fprintf('  n    mut_x       mut_y  \n');

    L = 2*0.00635; % distance ultrasonic pulse travels (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        for n=1:1
            % Kalman filter prediction
            mubart = g * mutminus1;
            Sigmabart = G * Sigmatminus1 * G' + R;

            % Compute measurement function (h) and measurement Jacobian (H)
            h = [sqrt( (x(1,1) - mubart(1))^2 + (y(1,1) - mubart(2))^2 )
                sqrt( (x(2,1) - mubart(1))^2 + (y(2,1) - mubart(2))^2 )
                sqrt( (x(3,1) - mubart(1))^2 + (y(3,1) - mubart(2))^2 )
                sqrt( (x(4,1) - mubart(1))^2 + (y(4,1) - mubart(2))^2 ) ];
            Hn = [ ( mubart(1) - x(1,1) ) / h(1)     ( mubart(2) - y(1,1) ) / h(1)
                ( mubart(1) - x(2,1) ) / h(2)     ( mubart(2) - y(2,1) ) / h(2)
                ( mubart(1) - x(3,1) ) / h(3)     ( mubart(2) - y(3,1) ) / h(3)
                ( mubart(1) - x(4,1) ) / h(4)     ( mubart(2) - y(4,1) ) / h(4) ];

            % Compute Kalman gain
            Kn = Sigmabart * Hn' * inv( Hn * Sigmabart * Hn' + Q );

            % Get a measurement update using the simulated source
            for k = 1:n_sensors
                Zn(k,1) =  L/v0 * (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ;
                 y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-T0));
            end
            Zn = Zn .* (1 + 5e-7 .* randn(n_sensors,1)); % add noise to TOF measurements

            % Use COMSOL to lookup radius using measured TOF
            T = postinterp(fem, 'T',[ rx ; ry ; rz ], 'T', t(i))-T0;
            T = reshape(T,zn,zn);
```

```
            T = mean(T',2);
            TOF = L/v0 * (1 + xi * T);
            Zn = interp1( TOF, r, Zn, 'spline');

            % Limit radius to rmax
            if max(Zn) > rmax
                for Zi = 1:length(Zn)
                    if Zn(Zi) > rmax
                        Zn(Zi) = rmax;
                    end
                end
            end

            % Kalman filter correction
            mut = mubart + Kn * (Zn - h);
            Sigmat = (eye(2) - Kn * Hn) * Sigmabart;

            % Limit estimated source location to points on the plate
            if abs(mut(1)) > xmax
                mut(1) = xmax*sign(mut(1));
            end
            if abs(mut(2)) > ymax
                mut(2) = ymax*sign(mut(2));
            end

            Mu = [ Mu mut];
            Mubar = [Mubar mubart];
            Sigmatminus1 = Sigmat;
            mutminus1 = mut;

        end
        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end
```

```
save 'PulseEchoRadius_88.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Ultrasonic pulse time of flight measurement model

Normalize change in TOF - dG/G

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 6.6e-5; % sec
measurement_variance = 4.79e-10; % sec^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = linspace(0.00635, 0.00635, n);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = linspace(0.00635, 0.00635, n);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = linspace(0.00635, 0.00635, n);

% x(5,:) = linspace(-d, d, n);
% y(5,:) = linspace(-d, d, n);
% z(5,:) = linspace(0.00635, 0.00635, n);
%
% x(6,:) = linspace(-d, d, n);
% y(6,:) = linspace(d, -d, n);
% z(6,:) = linspace(0.00635, 0.00635, n);

% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];

% Define time steps to locate source (s)
```

```
t=[300:1:360];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % covariance matrix

% Define measurement model matrices
Q = measurement_variance * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.0
        0.0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_x      mut_y  \n');

    R_ij = 2*d; % distance between sensors (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        % Compute measurement function (h) and measurement Jacobian (H)
        h = [];
        Ht = [];
        for k = 1:n_sensors
            h(k,1) = xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0);
            Ht(k,1) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dx;
            Ht(k,2) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)+dy-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dy;

            % Get a measurement update using the simulated source
            Zt(k,1) =  xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ; y(k,:)-ys(j) ; ...
             z(k,:) ], 'T', t(i)))-T0);
        end

        Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

        % Kalman filter correction
```

```
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
        end

        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

% save 'OneWayTOF_88.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
24 March 2010

Ultrasonic pulse ellipse measurement model

Normalize TOF - dG/G

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = 2.32e-4; % sec
measurement_variance = 4.624e-9;%9.28e-4; % m^2
state_variance = 0.0001; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = linspace(0.00635, 0.00635, n);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = linspace(0.00635, 0.00635, n);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = linspace(0.00635, 0.00635, n);

% x(5,:) = linspace(-d, d, n);
% y(5,:) = linspace(-d, d, n);
% z(5,:) = linspace(0.00635, 0.00635, n);
%
% x(6,:) = linspace(-d, d, n);
% y(6,:) = linspace(d, -d, n);
% z(6,:) = linspace(0.00635, 0.00635, n);

% Define the contstants
R_ij = 2*d; % distance between sensors (m)
v0 = 5100; % reference sound speed (?? K)
dx = 0.0001;
dy = 0.0001;
```

```
% Define actual source location (m)
xs = [0.02 0.04 0.05 0.06];
ys = [0 0 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Define plate edges
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define b values for indirect measurements (ellipse parameter)
amax = 0.1;
b = linspace(0,amax,n);
bx = meshgrid(b);
bx = bx(:)';
by = linspace(-d, d, n);
by = meshgrid(by)';
by = by(:)';
bz = meshgrid(z(1,:));
bz = bz(:)';

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % covariance matrix

% Define measurement model matrices
% Q = 0.000204 * eye(n_sensors); % covariance matrix
% Q = 4.624e-9 * eye(n_sensors); % covariance matrix
Q = measurement_variance * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0
        0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_x      mut_y  \n');

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        % Compute measurement function (h) and measurement Jacobian (H)
        % h = a (ellipse parameter) one value (row) for each sensor pair
        % H = [ da/dx da/dy] one row for each sensor pair
        h = [];
        Ht = [];
        for k = 1:n_sensors
            r_iq = sqrt( (x(k,1) - mubart(1))^2 + (y(k,1) - mubart(2))^2 ) ;
```

```
        r_jq = sqrt( (x(k,n) - mubart(1))^2 + (y(k,n) - mubart(2))^2 ) ;
        h = [ h
            ( r_iq + r_jq ) / 2 ];
        Ht = [ Ht
            1/2*( ( mubart(1) - x(k,1) ) / r_iq + ( mubart(1) - x(k,n) ) / r_jq )
            1/2*( ( mubart(2) - y(k,1) ) / r_iq + ( mubart(2) - y(k,n) ) / r_jq )  ];

        % Get a measurement update using the simulated source
        Zt(k,1) =  xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ; y(k,:)-ys(j) ; z(k,:) ], ...
         'T', t(i)))-T0);
    end

    Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

    % Lookup TOF values in COMSOL for a range of b values
    % b is analogous to radius - orthogonal distance from US path between
    % sensors to source.
    % use a(b) when interpolating TOF to get the observed value for a
    T = postinterp(fem, 'T',[ bx ; by ; bz ], 'T', t(i))-T0;
    T = reshape(T,n,n);
    T = mean(T',2);
    TOF = xi * T;

    a = 1/2 * sqrt(R_ij^2 + 4 * b.^2);
    Zt = interp1( TOF, a, Zt, 'spline');

    % Limit a to amax
    if max(Zt) > amax
        for Zi = 1:length(Zt)
            if Zt(Zi) > amax
                Zt(Zi) = amax;
            end
        end
    end

    % Compute Kalman gain
    Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

    % Kalman filter correction
    mut = mubart + Kt * (Zt - h);
    Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

    % Limit estimated source location to points on the plate
    if abs(mut(1)) > xmax
        mut(1) = xmax*sign(mut(1));
    end
    if abs(mut(2)) > ymax
        mut(2) = ymax*sign(mut(2));
    end


    Mu = [ Mu mut];
    Mubar = [Mubar mubart];
    Sigmatminus1 = Sigmat;
    mutminus1 = mut;

    if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
        converged = [];
    else
        converged = 'unconverged';
    end
    fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
    dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

    walltime(i) = toc;
end

fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));
```

```
    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

% save 'OneWayEllipse_00.out' dat2 -ascii -tabs
```

```
%{
Sensitivity study for one-way pulse TOF
Mike Myers
8 Feb 2010

%}
close all
clc

% Define sensor locations (m)
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
[x,y,z] = meshgrid(-d:2*d/(n-1):d,0,0.00635);

% Define range of source locations to evaluate (m)
ds = 0.08;
ns = 60; % number of points in x and y directions
[xs,ys,zs] = meshgrid(-ds:2*ds/(ns-1):ds,-ds:2*ds/(ns-1):ds,0.00635);

% Set the delta values for obtaining derivatives
dx = 0.0001;
dy = 0.0001;

% build x, y, and z vectors of source locations
xt = ones(3600,1)*x - xs(:)*ones(1,100);
yt = ones(3600,1)*y - ys(:)*ones(1,100);
zt = ones(3600,1)*z;

% build x, y, and z vectors for getting Tavg, dTavg/dx, and dTavg/dy
xt = [xt; xt + dx; xt];
yt = [yt; yt; yt+dy];
zt = [zt; zt; zt];

% simulation time t for evaluation (s)
t = [450:1:450];

for i=1:length(t)

    % Initialize variables
    T =[];
    dTavg_dx = [];
    dTavg_dy = [];

    % Get temperature values from COMSOL
    T = postinterp(fem, 'T-Tinf',[xt(:)' ; yt(:)' ; zt(:)' ], 'T', t(i));

    % Reshape matrix into one row for each sensor path set
    T = reshape(T, 3*ns*ns, n);

    % Get average temperature for each sensor path set
    T = mean(T,2);

    % Put average temperatures into square matrix
    Tavg = reshape(T(1:ns*ns), ns, ns)';

    % Compute derivatives
    dTavg_dx = (reshape(T(ns*ns+1:2*ns*ns), ns, ns)' - Tavg) / dx;
    dTavg_dy = (reshape(T(2*ns*ns+1:3*ns*ns), ns, ns)' - Tavg) / dy;

    % Compute the gradient of the average temperature
    gradTavg = sqrt(dTavg_dx.^2 + dTavg_dy.^2);

    % Plot the figures
%     figure(1)
%     contourf(xs,ys,Tavg',10);
%     hold on
%     plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
```

```
%     hold off
%     colorbar('location','eastoutside')
%     xlabel('x (m)','FontSize',12)
%     ylabel('y (m)','FontSize',12)
%     title(['\theta_{avg} [K] between fixed sensors for
%    different source locatons, t=',num2str(t(i)), ' sec']);
%
%     figure(2)
%     contourf(xs,ys,dTavg_dx',10);
%     hold on
%     plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
%     hold off
%     colorbar('location','eastoutside')
%     xlabel('x (m)','FontSize',12)
%     ylabel('y (m)','FontSize',12)
%     title(['\partial \theta_{avg} / \partial x [K/m] for fixed sensors and
%     different source locatons, t=',num2str(t(i)), ' sec']);
%
%     figure(3)
%     contourf(xs,ys,dTavg_dy',10);
%     hold on
%     plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
%     hold off
%     colorbar('location','eastoutside')
%     xlabel('x (m)','FontSize',12)
%     ylabel('y (m)','FontSize',12)
%     title(['\partial \theta_{avg} / \partial y [K/m] for fixed sensors and
%     different source locatons, t=',num2str(t(i)), ' sec']);
%
    figure(4)
    contour3(xs,ys,gradTavg',1000);
    hold on
    plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
% contour3(xs*0.0001,ys*0.0001,Tavg,10);
    colorbar('location','eastoutside')
    xlabel('x (m)','FontSize',12)
    ylabel('y (m)','FontSize',12)
    zlabel('S_{xy} (K/m)','FontSize',12)
%     title(['Location Sensitivity S_{xy} for fixed sensors and different
%     source locatons, t=',num2str(t(i)), ' sec']);
    zlim([0 1600]);
    xlim([-ds ds]);
    ylim([-ds ds]);
    caxis([0 1500]);
    view(84,48)

%     saveas(gcf,['grad_t',num2str(t(i)),'.png'], 'png');

%     figure(5)
%     contourf(xs,ys,gradTavg',10);
%     hold on
%     plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
%     hold off
%     colorbar('location','eastoutside')
%     xlabel('x (m)','FontSize',12)
%     ylabel('y (m)','FontSize',12)
%     title(['|\nabla \theta_{avg}| for fixed sensors and different source
%     locatons, t=',num2str(t(i)), ' sec']);
%
end
```

```
%{
Plot temperature profile in plate for a range of times
Mike Myers
1 Feb 2010

%}

close all
clc
% clear T

% Define number of points in x and y directions to evaluate temperature
n = 60;

% simulation time t for evaluation (s)
t = [320:1:320];

[x,y,z] = meshgrid(-0.15:0.3/(n-1):0.15,-0.15:0.3/(n-1):0.15,0.00635);


for ti=1:length(t)

    % retrieve temperatures for all defined points in the plate
    T = postinterp(fem, 'T-Tinf',[x(:)' ; y(:)' ; z(:)' ], 'T', t(ti));
    T = reshape(T,60,60);

    figure(1)
    contour3(x, y, T, 1000);
    %     hold on
    % plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
    colorbar('location','eastoutside')
    xlabel('x (m)','FontSize',12)
    ylabel('y (m)','FontSize',12)
    zlabel('\theta (K)','FontSize',12)
    title(['Temperature Change ( \theta ), t=',num2str(t(ti)), ' sec'],'FontSize',12);
    zlim([0 100]);
    caxis([0 50]);

%     saveas(gcf,['temp_t',num2str(t(ti)),'.png'], 'png');

    % old code - way slower method!
    %  tic
    %     % Compute the temperature across the plate
    %     % location
    %     for i=1:length(x)
    %         for j=1:length(y)
    %             T(i,j) = postinterp(fem, 'T-Tinf',[x(i) ; y(j) ; 0.00635 ], 'T', t(ti));
    %         end
    %     end
    %     toc

    %     % Plot the figures
    %     figure(1)
    %     contour3(x, y, T', 1000);
    % %     hold on
    %     % plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
    %     colorbar('location','eastoutside')
    %     xlabel('x (m)','FontSize',12)
    %     ylabel('y (m)','FontSize',12)
    %     zlabel('\theta (K)','FontSize',12)
    %     title(['Temperature Change ( \theta ), t=',num2str(t(ti)), ' sec'],'FontSize',12);
    %     zlim([0 100]);
    %     caxis([0 50]);
    %

end
```

```
%{
Plot temperature profile in plate for a range of times
Mike Myers
1 Feb 2010

%}

close all
clc
% clear T

load('RedColormap','mycmap'); % Load save colormap that works well for B&W printing

% Define range of locations to evaluate (m)
x = linspace(-0.1,0.1,100);
y = linspace(-0.1,0.1,100);

x1 = [-0.04 0.04];
y1 = [0 0];
% y2 = [-0.01 -0.01];

% Set the time to evaluate temperatures (s)

for t=320:320

    % Compute the temperature across the plate
    % location
%     for i=1:length(x)
%         for j=1:length(y)
%             T(i,j) = postinterp(fem, 'T-Tinf',[x(i) ; y(j) ; 0.00635 ], 'T', t);
%         end
%     end

    % Plot the figures

    figure(1)
    contourf(x+0.0, y+0.0, T', 10);
    hold on
    plot([x1(1),x1(2)],[y1(1),y1(2)],'ks-','Linewidth',2)
%     plot([x1(1),x1(2)],[y2(1),y2(2)],'ks-','Linewidth',2)
    colorbar('location','eastoutside')
    xlabel('x (m)','FontSize',12)
    ylabel('y (m)','FontSize',12)
%     title(['Temperature Change ( \theta ), t=',num2str(t), ' sec'],'FontSize',12);
    xlim([-0.05 0.05]);
    ylim([-0.05 0.05]);
    set(gcf,'Colormap',mycmap)

    %     saveas(gcf,['temp_t',num2str(t),'.pdf'], 'pdf');

end
```

```
%{
Plot temperature profile in plate for a range of times
Mike Myers
1 Feb 2010

%}

close all
clc
clear T

% Define range of locations to evaluate (m)
x = linspace(-0.15,0.15,600);
y = linspace(0,0,600);
z = linspace(0.00635,0.00635,600);

% Set the time to evaluate temperatures (s)

t = [320 450 600];

% for t=302:302

    % Compute the temperature across the plate
    % location
    T = postinterp(fem, 'T-Tinf',[x ; y ; z ], 'T', t);

    % Plot the figures

    figure(1)
    plot(x, T(1,:)','b','Linewidth',2);
    hold on
    grid on
    plot(x, T(2,:)','r','Linewidth',2);
    plot(x, T(3,:)','g','Linewidth',2);

    % plot([x(1),x(n)],[y(1),y(n)],'ks-','Linewidth',2)
    xlabel('x (m)','FontSize',12)
    ylabel('\theta (K)','FontSize',12)
    title(['Temperature Change ( \theta )'],'FontSize',12);
    xlim([-0.15 0.15]);
    ylim([0 150]);
    legend('320 sec','450 sec', '600 sec');
    %     saveas(gcf,['tempX_t',num2str(t),'.pdf'], 'pdf');

% end
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
26 June 2010

One-way ultrasonic pulse time of flight measurement model
using one-way experiment measurements

Normalize TOF - G/G0

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = [6e-5];% 12e-5 18e-5];%6.6e-5; % non-dimensional
% measurement_variance = ((sensor_noise./3).^2)/1000;%4.79e-10;%6e-7;%2e-8; % non-dimensional
measurement_variance = 4e-7 * [1 1 1 1 1 1 1 1 1 1];
state_variance = 0.0001; % m^2

% load experiment data (t in column 1, Gt/Ginit in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
%  (plate temp before heating applied) (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature (for computing
%  average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);
% n_bounces = 1;
% y_length = 2 * d / (n_bounces + 1); % length of one bounce
% % Compute z values for a bouncing signal
% for i = 1:n
%     z(1,i) = 0.00635/y_length * ((y(1,i)+d) - floor((y(1,i)+d)/y_length)*y_length);
%     if mod(floor(y(1,i)/y_length), 2);
%         z(1,i) = 0.00635 - z(1,i); % path is downward toward z=0 so subtract from plate thickness
%     end
% end

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = z(1,:);
```

```
x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = z(1,:);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = z(1,:);

% Define actual source location (m)
xs = 0.02;%[0.02 0.04 0.06];
ys = 0;%[0 0 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % covariance matrix

datnoise = [];

for sni = 1:length(sensor_noise)

    % Define measurement model matrices
Q = measurement_variance(sni) * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.0
        0.0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_x      mut_y  \n');

    R_ij = 2*d; % distance between sensors (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        % Compute measurement function (h) and measurement Jacobian (H)
        h = [];
        Ht = [];
        for k = 1:n_sensors
```

```
            h(k,1) = 1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit);
            Ht(k,1) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - h(k,1)) / dx;
            Ht(k,2) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)+dy-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - h(k,1)) / dy;

            % Get a measurement update using the simulated source
            %             Zt(k,1) =  1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ;
            % y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-Tinit);


            %             h(k,1) = xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ;
            % y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0);
            %             Ht(k,1) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ;
            % y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dx;
            %             Ht(k,2) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ;
            % y(k,:)+dy-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dy;
            %
            %             % Get a measurement update using the simulated source
                        Zt(k,1) =  1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ; ...
                         y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-Tinit);
        end

                Zt = Zt .* (1 + sensor_noise(sni) .* randn(n_sensors,1)); % add noise to TOF measurements

%         if xs(j)==0.02
%             Zt = [measurements(i+t(1)+1,5),measurements(i+t(1)+1,4),
measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
%         else if xs(j)==0.04
%                 Zt = [measurements(i+t(1)+1,6),measurements(i+t(1)+1,4),
measurements(i+t(1)+1,2),measurements(i+t(1)+1,4)]';
%             else if xs(j)==0.06
%                     Zt = [measurements(i+t(1)+1,7),measurements(i+t(1)+1,4),
measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
%                 end
%             end
%         end
        %         h
        %         Ht
        %         Zt

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

        % Kalman filter correction
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
        end

        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
```

```
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end
datnoise = [datnoise ; dat2];
end

% save 'EKF_TOF.out' dat2 -ascii -tabs
```

```
%{
Extended Kalman Filter for heat source localization
Mike Myers
26 June 2010

Ultrasonic pulse ellipse measurement model

Normalize TOF - G/G0

%}

close all
clc
x = [];
y = [];
z = [];

sensor_noise = 6e-5; % non-dimensional
measurement_variance = 3.19e-10;
state_variance = 0.0001; % m^2

% load experiment data (t in column 1, Gt/Ginit in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
(plate temp before heating applied) (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = z(1,:);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = z(1,:);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = z(1,:);

% Define the contstants
R_ij = 2*d; % distance between sensors (m)
v0 = 5100; % reference sound speed (?? K)
```

```
dx = 0.0001;
dy = 0.0001;

% Define actual source location (m)
xs = 0.02;%[0.02 0.04 0.06];
ys = 0;%[0 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Define plate edges
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define b values for indirect measurements (ellipse parameter)
amax = 0.1;
b = linspace(0,amax,n);
bx = meshgrid(b);
bx = bx(:)';
by = linspace(-d, d, n);
by = meshgrid(by)';
by = by(:)';
bz = meshgrid(z(1,:));
bz = bz(:)';

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
R = state_variance * eye(2); % covariance matrix

for sni = 1:length(sensor_noise)

Q = measurement_variance(sni) * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.0
        0.0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_x      mut_y  \n');

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + R;

        % Compute measurement function (h) and measurement Jacobian (H)
        % h = a (ellipse parameter) one value (row) for each sensor pair
        % H = [ da/dx da/dy] one row for each sensor pair
        h = [];
        Ht = [];
        for k = 1:n_sensors
```

```
            r_iq = sqrt( (x(k,1) - mubart(1))^2 + (y(k,1) - mubart(2))^2 ) ;
            r_jq = sqrt( (x(k,n) - mubart(1))^2 + (y(k,n) - mubart(2))^2 ) ;
            h = [ h
                ( r_iq + r_jq ) / 2 ];
            Ht = [ Ht
                1/2*( ( mubart(1) - x(k,1) ) / r_iq + ( mubart(1) - x(k,n) ) / r_jq )
                 1/2*( ( mubart(2) - y(k,1) ) / r_iq + ( mubart(2) - y(k,n) ) / r_jq )  ];

            % Get a measurement update using the simulated source
%             Zt(k,1) =  1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ;
y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-Tinit);
        end

%         Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements
        if xs(j)==0.02
            Zt = [measurements(i+t(1)+1,5),measurements(i+t(1)+1,4),...
             measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
        else if xs(j)==0.04
                Zt = [measurements(i+t(1)+1,6),measurements(i+t(1)+1,4),...
                 measurements(i+t(1)+1,2),measurements(i+t(1)+1,4)]';
            else if xs(j)==0.06
                    Zt = [measurements(i+t(1)+1,7),measurements(i+t(1)+1,4),...
                     measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
                end
            end
        end

        % Limit minimum measurement to 1 (temperature cannot be less than Tinit)
        for Zi = 1:length(Zt)
            if Zt(Zi) < 1
                Zt(Zi) = 1;
            end
        end

                % Lookup TOF values in COMSOL for a range of b values
        % b is analogous to radius - orthogonal distance from US path between
        % sensors to source.
        % use a(b) when interpolating TOF to get the observed value for a
        T = postinterp(fem, 'T',[ bx ; by ; bz ], 'T', t(i))-Tinit;
        T = reshape(T,n,n);
        T = mean(T',2);
        TOF = 1 + xi * T;

        a = 1/2 * sqrt(R_ij^2 + 4 * b.^2);
        Zt = interp1( TOF, a, Zt, 'spline');

        % Limit a to amax
        if max(Zt) > amax
            for Zi = 1:length(Zt)
                if Zt(Zi) > amax
                    Zt(Zi) = amax;
                end
            end
        end

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Q );

        % Kalman filter correction
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
```

```
        end


        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)];

        walltime(i) = toc;
    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end
end

% save 'EKF_Ellipse_88.out' dat2 -ascii -tabs
```

```
%{
Least squares routine using time of flight
to locate heating source using COMSOL model
and data from one-way pulse experiment.

Mike Myers
11 Jan 2011

%}

close all;
% clear all;
clc;

x = [];
y = [];
z = [];

% load experiment data (t in column 1, Gt/Ginit in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
(plate temp before heating applied) (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = z(1,:);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = z(1,:);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = z(1,:);

% Define actual source location (m)
xs = [0.02];% 0.04 0.06];
ys = [0];% 0 0];

% Define time steps to locate source (s)
t=[300:1:360];
```

```
% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

for j=1:length(xs)

    % Initialize least squares parameters
    X0 = [0.08
        0.08]; % initial guess for source location
    X = X0; % history of states
    Xtminus1 = X0;
    G=[];
    B=[];
    Z=[];
    walltime = [];

    fprintf('  t      x      y  \n');

    R_ij = 2*d; % distance between sensors (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Least squares predicted state
        Xt = Xtminus1;

        % Compute sensitivity matrix B
        G = [];
        B = [];
        for k = 1:n_sensors
            G(k,1) = 1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-Xt(1) ; ...
             y(k,:)-Xt(2) ; z(k,:) ], 'T', t(i)))-Tinit);
            B(k,1) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-Xt(1) ; ...
             y(k,:)-Xt(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - G(k,1)) / dx;
            B(k,2) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-Xt(1) ; ...
             y(k,:)+dy-Xt(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - G(k,1)) / dy;
        end

        if xs(j)==0.02
            Z = [measurements(i+t(1)+1,5),measurements(i+t(1)+1,4),...
             measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
        else if xs(j)==0.04
                Z = [measurements(i+t(1)+1,6),measurements(i+t(1)+1,4),...
                 measurements(i+t(1)+1,2),measurements(i+t(1)+1,4)]';
            else if xs(j)==0.06
                    Z = [measurements(i+t(1)+1,7),measurements(i+t(1)+1,4),...
                     measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
                end
            end
        end

        % Least squares correction
        deltaX = inv(B'*B)*B'*(Z - G);
        maxchange = 0.01; % (m) prevent really large changes to the state
        if abs(deltaX(1))>maxchange
            deltaX(1)=maxchange*sign(deltaX(1));
        end
        if abs(deltaX(2))>maxchange
            deltaX(2)=maxchange*sign(deltaX(2));
        end
        Xt = Xt + deltaX;

        % Limit estimated source location to points on the plate
```

```
        if abs(Xt(1)) > xmax
            Xt(1) = xmax*sign(Xt(1));
        end
        if abs(Xt(2)) > ymax
            Xt(2) = ymax*sign(Xt(2));
        end

        X = [ X Xt ];
        Xtminus1 = Xt;

        if abs(Xt(1)-xs(j)) < 0.001 && abs(Xt(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), Xt(1), Xt(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) Xt(1) Xt(2)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(X)

    iteration = linspace(0,length(X)-1,length(X));

    figure
    subplot(2,1,1)
    plot(iteration, X(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, X(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

%save 'LS_TOF_88.out' dat2 -ascii -tabs
```

```
%{
Least squares routine using ellipse model
to locate heating source using COMSOL model
and data from one-way pulse experiment.

Mike Myers
11 Jan 2011

%}

close all;
% clear all;
clc;

x = [];
y = [];
z = [];

% load experiment data (t in column 1, Gt/Ginit in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
(plate temp before heating applied) (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = z(1,:);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = z(1,:);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = z(1,:);

% Define actual source location (m)
xs = [0.02];% 0.04 0.06];
ys = [0];% 0 0];

% Define time steps to locate source (s)
t=[300:1:360];
```

```
% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Initialize saved data
dat2 = zeros(length(t)*length(xs),5);

% Define b values for indirect measurements (ellipse parameter)
amax = 0.1;
b = linspace(0,amax,n);
bx = meshgrid(b);
bx = bx(:)';
by = linspace(-d, d, n);
by = meshgrid(by)';
by = by(:)';
bz = meshgrid(z(1,:));
bz = bz(:)';

for j=1:length(xs)

    % Initialize least squares parameters
    X0 = [0.08
        0.08]; % initial guess for source location
    X = X0; % history of states
    Xtminus1 = X0;
    G=[];
    B=[];
    Z=[];
    walltime = [];

    fprintf('  t      x      y  \n');

    R_ij = 2*d; % distance between sensors (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Least squares predicted state
        Xt = Xtminus1;

        % Compute sensitivity matrix B
        G = [];
        B = [];
        for k = 1:n_sensors
            r_iq = sqrt( (x(k,1) - Xt(1))^2 + (y(k,1) - Xt(2))^2 ) ;
            r_jq = sqrt( (x(k,n) - Xt(1))^2 + (y(k,n) - Xt(2))^2 ) ;
            G = [ G
                ( r_iq + r_jq ) / 2 ];
            B = [ B
                1/2*( ( Xt(1) - x(k,1) ) / r_iq + ( Xt(1) - x(k,n) ) / r_jq )
                 1/2*( ( Xt(2) - y(k,1) ) / r_iq + ( Xt(2) - y(k,n) ) / r_jq )  ];
        end

        if xs(j)==0.02
            Z = [measurements(i+t(1)+1,5),measurements(i+t(1)+1,4),...
             measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
        else if xs(j)==0.04
                Z = [measurements(i+t(1)+1,6),measurements(i+t(1)+1,4),...
                 measurements(i+t(1)+1,2),measurements(i+t(1)+1,4)]';
            else if xs(j)==0.06
                    Z = [measurements(i+t(1)+1,7),measurements(i+t(1)+1,4),...
                     measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
                end
            end
        end
```

```
    % Limit minimum measurement to 1 (temperature cannot be less than Tinit)
    for Zi = 1:length(Z)
        if Z(Zi) < 1
            Z(Zi) = 1;
        end
    end

    % Lookup TOF values in COMSOL for a range of b values
    % b is analogous to radius - orthogonal distance from US path between
    % sensors to source.
    % use a(b) when interpolating TOF to get the observed value for a
    T = postinterp(fem, 'T',[ bx ; by ; bz ], 'T', t(i))-Tinit;
    T = reshape(T,n,n);
    T = mean(T',2);
    TOF = 1 + xi * T;

    a = 1/2 * sqrt(R_ij^2 + 4 * b.^2);
    Z = interp1( TOF, a, Z, 'spline');

    % Limit a to amax
    if max(Z) > amax
        for Zi = 1:length(Z)
            if Z(Zi) > amax
                Z(Zi) = amax;
            end
        end
    end


    % Least squares correction
    deltaX = inv(B'*B)*B'*(Z - G);
    maxchange = 0.01; % (m) prevent really large changes to the state
    if abs(deltaX(1))>maxchange
        deltaX(1)=maxchange*sign(deltaX(1));
    end
    if abs(deltaX(2))>maxchange
        deltaX(2)=maxchange*sign(deltaX(2));
    end
    Xt = Xt + deltaX;

    % Limit estimated source location to points on the plate
    if abs(Xt(1)) > xmax
        Xt(1) = xmax*sign(Xt(1));
    end
    if abs(Xt(2)) > ymax
        Xt(2) = ymax*sign(Xt(2));
    end

    X = [ X Xt ];
    Xtminus1 = Xt;

    if abs(Xt(1)-xs(j)) < 0.001 && abs(Xt(2)-ys(j)) < 0.001
        converged = [];
    else
        converged = 'unconverged';
    end
    fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), Xt(1), Xt(2))
    dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) Xt(1) Xt(2)];

    walltime(i) = toc;

end

fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

% convergence_plot(X)

iteration = linspace(0,length(X)-1,length(X));
```

```
    figure
    subplot(2,1,1)
    plot(iteration, X(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, X(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

end

%save 'LS_ellipse_88.out' dat2 -ascii -tabs
```

```
%{
Particle filter to locate a high heat flux point source in a flat plate
Mike Myers
11 Nov 2010

Generates 40 uniformly distributed particles on the
plate, assigns a weight to each particle based
on the expected sensor readings for the particle, normalizes the weights,
then resamples.

%}

close all
clc

% load experiment data (t in column 1, Gt/G0 in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
 (plate temp before heating applied) (K)

% Define sensor locations (m)
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
 (for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);

x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = linspace(0.00635, 0.00635, n);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = linspace(0.00635, 0.00635, n);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = linspace(0.00635, 0.00635, n);

% Define true source location (m)
xs = [0.02];% 0.04 0.06];
ys = [0];% 0 0];

% Define time steps to locate source (s)
t=[300:1:360];

% Distribute M particles across plate
M=40; % Number of particles (40 for 8x8, 90 for 12x12)

% Plate boundaries
xx = [-0.305; -0.305; 0.305; 0.305; -0.305];
yy = [-0.1575; 0.1575; 0.1575; -0.1575; -0.1575];
```

```
predicted_location = zeros(length(t)*length(xs),5);

for j=1:length(xs)

    % Randomly distribute (uniform) the particles
    rand('state',sum(100*clock))
    particle = rand(M,2) * [0.12 0; 0 0.12]; % restrict particles to portion of plate where we can
    %                                                                    get expected sensor readings from model
    particle(:,1) = particle(:,1) - 0.04;%0.265;
    particle(:,2) = particle(:,2) - 0.04;%0.1175;

    gain = 6e4;

    E= zeros(length(t),M);

    mu0 = [0.0
        0.0];
    Mu = mu0;
    walltime = zeros(1,length(t));

    fprintf('  t         x         y  \n');

    for i = 1:length(t)

        tic

        %         Plot the plate with all particles in their current state
%         figure
%         plot(xx,yy,'-','Linewidth',2)
%         hold on;
%         for k = 1:n_sensors
%             plot(x(k,:),y(k,:),'-g','Linewidth',2)
%         end
%         plot(particle(:,1),particle(:,2),'.r','Linewidth',2)
%         axis([-0.08 0.08 -0.08 0.08]);
%         xlabel('X Position (m)','FontSize',12)
%         ylabel('Y Position (m)','FontSize',12)
%         title(['Distribution of particles at ',num2str(t(i)),' sec'],'FontSize',12)
        %         print('-depsc','-tiff','-r300',['images\picture',num2str(2*(i-1)+1)])

        % Add random Gaussian noise to particle position ~N(0,0.01)
        particle_old = particle;
        particle(:,1) = particle(:,1) + (rand(M,1)-.5)*0.01;
        particle(:,2) = particle(:,2) + (rand(M,1)-.5)*0.01;

        % Plot the plate with old particles and new particles after adding noise
        %         figure
        %         plot(xx,yy,'-','Linewidth',2)
        %         hold on;
        %         for k = 1:n_sensors
        %             plot(x(k,:),y(k,:),'-g','Linewidth',2)
        %         end
        %         plot(particle_old(:,1),particle_old(:,2),'.r')
        %         plot(particle(:,1),particle(:,2),'.r','Linewidth',2)
        %         axis([-0.08 0.08 -0.08 0.08]);
        %         xlabel('X Position (m)','FontSize',12)
        %         ylabel('Y Position (m)','FontSize',12)
        %         title(['Distribution of particles at ',num2str(t(i)),' sec'],'FontSize',12)
        %         print('-depsc','-tiff','-r300',['images\picture',num2str(2*i)])

        % Take expected measurements for each particle
        Z = zeros(M, n_sensors);
        for p = 1:M
            for k = 1:n_sensors
                Z(p,k) =  (1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-particle(p,1) ;
                 y(k,:)-particle(p,2) ; z(k,:) ], 'T', t(i)))-Tinit));
            end
```

```
    end

    % Get measurement using true position
    Ztrue = zeros(1,n_sensors);
    switch(xs(j))
        case 0.0
            Ztrue = [measurements(i+t(1)+1,4) measurements(i+t(1)+1,4)  ...
             measurements(i+t(1)+1,4)  measurements(i+t(1)+1,4)];
        case 0.02
            Ztrue = [measurements(i+t(1)+1,5) measurements(i+t(1)+1,4)  ...
             measurements(i+t(1)+1,3)  measurements(i+t(1)+1,4)];
        case 0.04
            Ztrue = [measurements(i+t(1)+1,6)  measurements(i+t(1)+1,4)  ...
             measurements(i+t(1)+1,2)  measurements(i+t(1)+1,4)];
        case 0.06
            Ztrue = [measurements(i+t(1)+1,7)  measurements(i+t(1)+1,4)  ...
             measurements(i+t(1)+1,3)  measurements(i+t(1)+1,4)];
    end

    % Evaluate each particle and weight it according to N(Ztrue,I)
    Sigma = eye(n_sensors);
    Sigma_inv = inv( Sigma );
    for p=1:M
        E(i,p) = exp( -1/2 * ((Z(p,:) - Ztrue)*gain) * Sigma_inv * ((Z(p,:) - Ztrue)*gain)' );...
        particle(p,3) = det(2 * pi * Sigma) ^ (-1/2) * exp( -1/2 * ((Z(p,:) - ...
         Ztrue)*gain) * Sigma_inv * ((Z(p,:) - Ztrue)*gain)' );
    end
    particle(isnan(particle)) = 0; % replace all occurrences of NaN with 0

    % Normalize weights into bins from 0 to 1
    s = sum(particle(:,3));
    particle(1,4) = particle(1,3)/s;
    for p=2:M
        particle(p,4) = particle(p,3)/s + particle(p-1,4);
    end

    particle_old = particle;

    % Generate M random numbers with normal distribution
    data = rand(M,1);
    data = sortrows(data);

    % Use histogram count to resample best particles
    [n,bin] = histc(data, [0;particle(:,4)]);
    particle = particle(bin,1:2);

    mut = [mean(particle(:,1))
        mean(particle(:,2))];
    Mu = [Mu mut];
    predicted_location((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1)  mut(2)];
    if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
        converged = [];
    else
        converged = 'unconverged';
    end
    fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))

    walltime(i) = toc;

end

%     figure
%     plot(particle_old(:,1),particle_old(:,4),'-bs')

fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

% Plot the plate after last iteration
figure
```

```
    plot(xx,yy,'-','Linewidth',2)
    hold on;
    for k = 1:n_sensors
        plot(x(k,:),y(k,:),'-g','Linewidth',2)
    end
    plot(particle(:,1),particle(:,2),'.r')
    axis([-0.08 0.08 -0.08 0.08]);
    xlabel('X Position (m)','FontSize',12)
    ylabel('Y Position (m)','FontSize',12)
    title(['Distribution of particles at end ',num2str(t(i)),' sec'],'FontSize',12)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(iteration, Mu(1,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('x location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    subplot(2,1,2)
    plot(iteration, Mu(2,:)*100,'-r','Linewidth',3)
    grid on
    xlabel('Iteration','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    fprintf('\n Average of all particles is (%2.1f cm, %2.1f cm)\n\n',
     mean(particle(:,1))*100, mean(particle(:,2))*100);

end

%save 'PF_12x12.out' predicted_location -ascii -tabs
```

```
%{
Adaptive Extended Kalman Filter for heat source localization
Mike Myers
16 Sep 2011

One-way ultrasonic pulse time of flight measurement model
using one-way experiment measurements

Normalize TOF - G/G0

Adaptive covariance

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = [6e-5];% 12e-5 18e-5];%6.6e-5; % non-dimensional
% measurement_variance = ((sensor_noise./3).^2)/1000;%4.79e-10;
%6e-7;%2e-8; % non-dimensional
measurement_variance = 4e-7 * [1 1 1 1 1 1 1 1 1 1]; % 4e-7
state_variance = 0.00001; %0.0001 m^2

% load experiment data (t in column 1, Gt/Ginit in columns 2-7)
% column 2 - source at (0,0cm)
% column 3 - source at (0,2cm)
% column 4 - source at (0,4cm)
% column 5 - source at (0,6cm)
% column 6 - source at (0,8cm)
% column 7 - source at (0,10cm)
load measurements.txt

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit =  postinterp(fem, 'T',[0 ; 0 ; 0 ], 'T', 290); % initialization temperature
 (plate temp before heating applied) (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
 (for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d, -d, n);
y(1,:) = linspace(-d, d, n);
z(1,:) = linspace(0.00635, 0.00635, n);
% n_bounces = 1;
% y_length = 2 * d / (n_bounces + 1); % length of one bounce
% % Compute z values for a bouncing signal
% for i = 1:n
%     z(1,i) = 0.00635/y_length * ((y(1,i)+d) - floor((y(1,i)+d)/y_length)*y_length);
%     if mod(floor(y(1,i)/y_length), 2);
%         z(1,i) = 0.00635 - z(1,i); % path is downward toward z=0 so subtract from plate thickness
%     end
% end
```

```
x(2,:) = linspace(-d, d, n);
y(2,:) = linspace(d, d, n);
z(2,:) = z(1,:);

x(3,:) = linspace(d, d, n);
y(3,:) = linspace(d,-d,n);
z(3,:) = z(1,:);

x(4,:) = linspace(d, -d, n);
y(4,:) = linspace(-d, -d, n);
z(4,:) = z(1,:);

% Define actual source location (m)
xs = 0.02;%[0.02 0.04 0.06];
ys = 0;%[0 0 0 0];

% Define time steps to locate source (s)
t=[300:1:340];

% Define the maximum source position
xmax = 0.1;
ymax = 0.1;

% Define the maximum desired convergence rate (m/s)
DeltaXt_limit = [0.001
    0.001];

% Initialize saved data
dat2 = zeros(length(t)*length(xs),15);

% Define state transition matrices
g = [1 0
    0 1];
G = eye(2); % state Jacobian
Qt = state_variance * eye(2); % covariance matrix
Qt_history = Qt(1,1);

datnoise = [];

for sni = 1:length(sensor_noise)

    % Define measurement model matrices
Rt = measurement_variance(sni) * eye(n_sensors); % covariance matrix

for j=1:length(xs)

    % Initialize EKF parameters
    mu0 = [0.0
        0.0];
    Sigma0 = 0;
%     Sigma0 = [1e-3 0
%         0 1e-3];
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];
    Sigmat_value = 0;
    Sigmat_value_minus1 = 0;
    Kt_history = [0 0 0 0 0 0 0 0];
    Mt = 2; % Myers gain

    fprintf('  t      mut_x      mut_y  \n');
```

```
    R_ij = 2*d; % distance between sensors (m)
    dx = 0.0001;
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + Qt;

        % Compute measurement function (h) and measurement Jacobian (H)
        h = [];
        Ht = [];
        for k = 1:n_sensors
            %         theta(j,1) = mean(postinterp(fem, 'T',[x(j,:)-mubart(1) ; ...
%             y(j,:)-mubart(2) ; z(j,:) ], 'T', t(i)))-T0;
            h(k,1) = 1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit);
            Ht(k,1) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ; ...
             y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - h(k,1)) / dx;
            Ht(k,2) = -((1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ; ...
             y(k,:)+dy-mubart(2) ; z(k,:) ], 'T', t(i)))-Tinit)) - h(k,1)) / dy;

            % Get a measurement update using the simulated source
            %             Zt(k,1) =  1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ;
            % y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-Tinit);


            %             h(k,1) = xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ;
            % y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0);
            %             Ht(k,1) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)+dx-mubart(1) ;
            % y(k,:)-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dx;
            %             Ht(k,2) = -((xi * (mean(postinterp(fem, 'T',[x(k,:)-mubart(1) ;
            % y(k,:)+dy-mubart(2) ; z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dy;
            %
            %             % Get a measurement update using the simulated source
                        Zt(k,1) =  1 + xi * (mean(postinterp(fem, 'T',[x(k,:)-xs(j) ; ...
                         y(k,:)-ys(j) ; z(k,:) ], 'T', t(i)))-Tinit);
        end

                Zt = Zt .* (1 + sensor_noise(sni) .* randn(n_sensors,1)); % add noise to TOF measurements

%         if xs(j)==0.02
%             Zt = [measurements(i+t(1)+1,5),measurements(i+t(1)+1,4), ...
% measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
%         else if xs(j)==0.04
%                 Zt = [measurements(i+t(1)+1,6),measurements(i+t(1)+1,4), ...
% measurements(i+t(1)+1,2),measurements(i+t(1)+1,4)]';
%             else if xs(j)==0.06
%                     Zt = [measurements(i+t(1)+1,7),measurements(i+t(1)+1,4), ...
% measurements(i+t(1)+1,3),measurements(i+t(1)+1,4)]';
%                 end
%             end
%         end
        %         h
        %         Ht
        %         Zt

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + Rt );

        % Kalman filter correction
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(2) - Kt * Ht) * Sigmabart;

        Sigmat_value = [Sigmat_value Sigmat(1,1)];
        delta_sigma2_tolerance = Sigmat_value(2)*0.1;
```

```
        Kt_history = [Kt_history
            Kt(1,:) Kt(2,:)];

        if abs(Sigmat(1,1) - Sigmatminus1(1,1)) > delta_sigma2_tolerance
            if abs(mut - mutminus1) < DeltaXt_limit
              Qt = Qt * Mt;
            else
               Qt = Qt / Mt;
            end
        end
%
%         if abs(mut - mutminus1) <= DeltaXt_limit
%             if (Sigmat(1,1) - Sigmatminus1(1,1)) > delta_sigma2_tolerance
%               Qt = Qt * Mt;
%             else
%                 if (Sigmat(1,1) - Sigmatminus1(1,1)) < 0
%                    Qt = Qt / Mt;
%                 end
%             end
%         else
%             Qt = Qt / Mt;
%         end

        Qt_history = [Qt_history Qt(1,1)];
%         Mt = 1;%Sigmat(1,1)/1e-4;
%         Mt = (Sigmat(1,1) - Sigmat_value_minus1)/state_variance*0.2+1;

        % Limit estimated source location to points on the plate
        if abs(mut(1)) > xmax
            mut(1) = xmax*sign(mut(1));
        end
        if abs(mut(2)) > ymax
            mut(2) = ymax*sign(mut(2));
        end

        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;
        Sigmat_value_minus1 = Sigmat_value(1,1);
        Mtminus1 = Mt;

        if abs(mut(1)-xs(j)) < 0.001 && abs(mut(2)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f   %3.6f    %3.6f ',converged,' \n'],t(i), mut(1), mut(2))
        dat2((j-1)*length(t)+i,:) = [xs(j) ys(j) t(i) mut(1) mut(2)
         Sigmat_value(i) Kt_history(i,:) Qt_history(i)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    subplot(2,1,1)
    plot(t, Mu(1,1:length(t))*100,'-r','Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('Estimated x location (cm)','FontSize',12)
    axis tight
```

```
    ylim([-2 8])

    subplot(2,1,2)
    plot(t, Mu(2,1:length(t))*100,'-r','Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('Estimated y location (cm)','FontSize',12)
    axis tight
    ylim([-2 8])

    figure
    plot(t, Sigmat_value(1:length(t)),'-r','Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('\Sigma variance value','FontSize',12)
    axis tight

    figure
    plot(t, Kt_history(1:length(t),:),'Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('Kalman Gain','FontSize',12)
    legend('(1,1)','(1,2)','(1,3)','(1,4)','(2,1)','(2,2)','(2,3)','(2,4)')
    axis tight

    figure
    plot(t, Qt_history(1:length(t)),'Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('Qt','FontSize',12)
    axis tight

end
datnoise = [datnoise ; dat2];
end

% save 'AEKF_Q1_5_R4_7_M2.out' dat2 -ascii -tabs
```

```
% Calculate the temperature in an infinite disk with finite
% thickness and spot heating (constant) located on the top and
% bottom at the axis.

% VP Kozlov, VS Adamchik and VN Lipovtsev, "Local heating of an
% unbounded orthotropic plate through a circular and annular
% domain," Journal of Engineering Physics and Thermophysics, v. 75,
% n. 5, pp. 1381-1391, 1989.

% This script implements equation 28 from the above reference
% compare to Comsol data to verify model

clear all;
close all;

q = 16e3;                % heat flux of spot heat source (W/m^2)
k = 14.6;                 % thermal conductivity (W/m K)
rho = 8000;              % density (kg/m^3)
c = 500;                  % specific heat (J/kg K)
%global alpha = 3.65e-6;   % k/(rho * c)  % thermal diffusivity (m^2/s)
global alpha = k/(rho * c);
global a = 0.092;        % radius of spot heat source (m)
global h = 0.002667;        % 1/2 plate thickness (m) which is equal to
Comsol plate thickness

% to compare to measurements ...
t_off = 100;

t = [0:10:290];
% t = [0,100,200,300,400,500];
% t = [0, 10, 20, 30, 40, 100];
% z = [h 0];
% z = linspace(0, 0.00635, 20);
z = 0.002667;
% thermocouple locations
r = [0.062 0.072 0.082 0.092 0.102 0.112 0.122];
% r = [0.05:0.01:0.2];
%r = 0;
% r = sqrt(2*r.^2);

% calculate the integrand for the radial integral
function y = Rint(xp, tau, rp)
    global alpha
    y = xp .* exp(-(xp.^2)/(4*alpha*tau)) .* besseli(0,rp*xp/(2*alpha*tau));
endfunction

% This is an approximation of Rint for large arguments of the
% bessel function
function y = Rint_large(xp, tau, rp)
    global alpha
    y = sqrt(alpha*tau*xp/(pi*rp)).*exp(-(xp.^2)/(4*alpha*tau) + xp*rp/(2*alpha*tau));
endfunction


% Calculate the integrand for the time integral
function y = Tint(xi, rp, zp)
    global alpha
    global h
    global a

    M = 10;
    m = [-M:M];

    for i = 1:length(xi)
        if (xi(i) == 0 || rp/(2*alpha*xi(i)) > 5000)
            y(i) = 0.0;
        else
            RI = quadl( "Rint", 0, a, [], [], xi(i), rp );
```

```
            theta = sum(exp(-h*h/alpha/xi(i)*(m-zp/(2*h)).^2))*h/sqrt(pi*alpha*xi(i));
            y(i) = exp(-rp*rp./(4*alpha*xi(i))) ./ xi(i) .* theta .* RI;
        end
    end

endfunction

% dat(:,1) = t';
j = 1;

% loop over points that I want to calculate (IMS locations)
for zi = 1:length(z)
    for ri = 1:length(r)
        printf( "\n%g %g %g %g\n", t(1), r(ri), z(zi), 0 );
        for ti = 2:length(t)
            T(ti) = q/(2*k*h) * quadl( "Tint", 0, t(ti), [], [], r(ri), z(zi) );
            if (t(ti) > t_off) % Use Duhamel's rule to turn heat flux "off"
                T(ti) = T(ti) - q/(2*k*h) * quadl( "Tint", 0, t(ti)-t_off, [], [], r(ri), z(zi) );
            end
            printf( "%g %g %g %g\n", t(ti), r(ri), z(zi), T(ti) );
            dat2(j,:) = [t(ti) r(ri) z(zi) T(ti)];
            j=j+1;
        end

%         fi = (zi-1)*length(r)+ri+1;
%         dat(:,fi) = T';
        % save "Kozlov.out" dat
    end
end

save "Kozlov.out" dat2
```

Code for Step Heating Source

```
Steps to create numerical model in COMSOL:

3D heat transfer, conduction, transient
Work plane z=0
Square 0.305 centered
Rectangle 0.19 x 0.14 centered
Free mesh parameters - rectangle: max element size 100e-3
Extrude 1 layer, 0.002667 m
Contstants: Tamb, T0, Tinf, Tr all equal to 297.5 [K]
Physics, subdomain: k=14.6 [W/mK], rho=8000 [kg/m^3], Cp=500 [J/kg K], init T(t0)=T0
Physics, boundary:
edges (1,2,5,12) h=3 [W/m^2 K], q=0 [W/m^2], Const=4.2525e-9
sides (3,4,9) h=12.9 [W/m^2 K], q=0 [W/m^2], Const=4.2525e-9
heated zone (8) h=12.9 [W/m^2 K], q=9.97e3 [W/m^2], Const=4.536e-8
Solver: Time 1, strict
```

```
%{
Parameter identification to find best match between the COMSOL
model and one step source experiment. Least squares
method used.

Mike Myers
4 Jan 2012

%}

close all;
clear all;
clc;

% Baseline parameters
q = 9.97e3; % heat flux [W/m^2]
h = 12.9;   % convection coefficient on sides [W/m^2K]

P = [q ; h ]; % Matrix of baseline parameters

% Import the data and normalize to the mean of the first 95 seconds
% before heater is turned on
% load run16
% run16 =  [A(:,1) - mean(A(1:95,1)), ...
%           A(:,2) - mean(A(1:95,2)), ...
%           A(:,3) - mean(A(1:95,3)), ...
%           A(:,4) - mean(A(1:95,4)), ...
%           A(:,5) - mean(A(1:95,5)), ...
%           A(:,6) - mean(A(1:95,6)), ...
%           A(:,7) - mean(A(1:95,7)), ...
%           A(:,8) - mean(A(1:95,8))];
load run17
experiment =  [A(:,1) - mean(A(1:95,1)); ...
          A(:,2) - mean(A(1:95,2)); ...
          A(:,3) - mean(A(1:95,3)); ...
          A(:,4) - mean(A(1:95,4)); ...
          A(:,5) - mean(A(1:95,5)); ...
          A(:,6) - mean(A(1:95,6)); ...
          A(:,7) - mean(A(1:95,7)); ...
          A(:,8) - mean(A(1:95,8))];
% load run18
% run18 =  [A(:,1) - mean(A(1:95,1)), ...
%           A(:,2) - mean(A(1:95,2)), ...
%           A(:,3) - mean(A(1:95,3)), ...
%           A(:,4) - mean(A(1:95,4)), ...
%           A(:,5) - mean(A(1:95,5)), ...
%           A(:,6) - mean(A(1:95,6)), ...
%           A(:,7) - mean(A(1:95,7)), ...
%           A(:,8) - mean(A(1:95,8))];
%
% load run19
% run19 =  [A(:,1) - mean(A(1:95,1)), ...
%           A(:,2) - mean(A(1:95,2)), ...
%           A(:,3) - mean(A(1:95,3)), ...
%           A(:,4) - mean(A(1:95,4)), ...
%           A(:,5) - mean(A(1:95,5)), ...
%           A(:,6) - mean(A(1:95,6)), ...
%           A(:,7) - mean(A(1:95,7)), ...
%           A(:,8) - mean(A(1:95,8))];
%
% run_average = mean(cat(3,run16,run17,run18,run19),3);
%
% plot(run16-run_average)
% figure
% plot(run17-run_average)
% figure
% plot(run18-run_average)
% figure
```

```
% plot(run19-run_average)

A = [importdata('q0997_h1290.txt')];
baseline = [A(1:1000,2) ; ...
            A(1001:2000,2) ; ...
            A(2001:3000,2) ; ...
            A(3001:4000,2) ; ...
            A(4001:5000,2) ; ...
            A(5001:6000,2) ; ...
            A(6001:7000,2) ; ...
            A(7001:8000,2) ];

A = [importdata('q0997+_h1290.txt')];
qt = [A(1:1000,2) ; ...
      A(1001:2000,2) ; ...
      A(2001:3000,2) ; ...
      A(3001:4000,2) ; ...
      A(4001:5000,2) ; ...
      A(5001:6000,2) ; ...
      A(6001:7000,2) ; ...
      A(7001:8000,2) ];

A = [importdata('q0997_h1290+.txt')];
ht = [A(1:1000,2) ; ...
      A(1001:2000,2) ; ...
      A(2001:3000,2) ; ...
      A(3001:4000,2) ; ...
      A(4001:5000,2) ; ...
      A(5001:6000,2) ; ...
      A(6001:7000,2) ; ...
      A(7001:8000,2) ];

plot(experiment(1:1000))
hold on
plot(baseline(1:1000),'r')
plot(experiment(1001:2000))
plot(experiment(2001:3000))
plot(experiment(3001:4000))
plot(baseline(1001:2000),'r')
plot(baseline(2001:3000),'r')
plot(baseline(3001:4000),'r')

figure
plot(experiment(4001:5000))
hold on
plot(baseline(4001:5000),'r')
plot(experiment(5001:6000))
plot(experiment(6001:7000))
plot(experiment(7001:8000))
plot(baseline(5001:6000),'r')
plot(baseline(6001:7000),'r')
plot(baseline(7001:8000),'r')

% Sensitivity matrix X (normalized so units are K)
dTdq = ( qt - baseline ) / (q * 0.001) * q;
dTdh = ( ht - baseline ) / (h * 0.001) * h;
X = [ dTdq , dTdh ];

% Compute the beta matrix
% beta = inv(X'*X)*X'*(experiment - baseline)  + 1 %
could also use X\(experiment - baseline(:,2))
beta = X\(experiment - baseline) + 1% Compute using Matlab's matrix division


new_q=beta(1)*q
new_h=beta(2)*h
```

```
%{
Get plate temperatures for a range of times from COMSOL
and store in a data file for use in computing sensitivities
Mike Myers
5 Jan 2012

%}

close all
clc
clear T Temp

% Define step size (delta x and delta y) to evaluate temperature
step = 0.001;
xmin = -0.05;
xmax = 0.05;
m = (xmax - xmin)/step + 1;
ymin = -0.15;
ymax = 0.15;
n = (ymax - ymin)/step + 1;
z = 0.002667;

% simulation time t for evaluation (s)
t = [[90:1:400] [405:5:1000]];

[x,y,z] = meshgrid(xmin:step:xmax,ymin:step:ymax,z);
T = zeros(length(t)*n:m);

for ti=1:length(t)

    % retrieve temperatures for all defined points in the plate
    Temp = postinterp(fem, 'T-Tinf',[x(:)' ; y(:)' ; z(:)' ], 'T', t(ti));
    Temp = reshape(Temp, n, m);
    T = [T;Temp];

end

save 'q997_hs129_k146+_x-5to5_y-15to15_z002667.txt' T -ascii -tabs
```

```
% Compute sensitivity to changes in step position (y) relative
% to a single transducer pair
% First pair is parallel to the step (horizontal)
% Second pair is normal to the step

% clear all
close all
clc

xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)

% Define step size (delta x and delta y) to evaluate temperature
step = 0.001;
xmin = -0.05;
xmax = 0.05;
n = (xmax - xmin)/step + 1; % columns
ymin = -0.15;
ymax = 0.15;
m = (ymax - ymin)/step + 1; % rows
z = 0.002667;

x = [xmin:step:xmax];
y = [ymin:step:ymax];

% simulation time t for evaluation (s)
t = [[90:1:400],[405:5:1000]];

% Import COMSOL data
% baseline = [importdata(['q997_hs129_k146_x-5to5_y-15to15_z002667.txt'])];

data_sets = size(baseline,1)/m; % number of timesteps in the data

% Compute time of flight
step = 0.001;

d = 0.08; % distance between sensors (m)
dpoints = d/step;
d2points = dpoints/2;

plate_center_x = (n-1)/2 + 1; % column number in the data for plate center

dG = zeros(81,1);

ic =  plate_center_x;

for ti = 1:length(t)
    [min_difference, array_position] = min(abs(t - t(ti))); ...
    plate_center_y = (array_position-1) * m + (m-1)/2 + 1;
    % row number in the data for plate center

    for j = 1:111
        % start at y = 0cm and go to -11cm
        jc = plate_center_y + 1 - j;
        dG(j,ti) = (xi/v0*...
            ( mean(baseline( jc, ic - d2points:ic + d2points )) ...
            - mean(baseline( jc - 1, ic - d2points:ic + d2points ))...
            )/0.001); % scaled sensitivity (y dG/dy)
    end
% figure
% plot([0:step:0.08],dG(:,ti),'-r');
end

% save 'Sy_vs_y_parallel.txt' dG -ascii -tabs


for ti = 1:length(t)
    [min_difference, array_position] = min(abs(t - t(ti))); ...
```

```
    plate_center_y = (array_position-1) * m + (m-1)/2 + 1;
    % row number in the data for plate center

    for j = 1:111
        % start sensor path center at y = 0cm and go to -11cm
        jc = plate_center_y + 1 - j;
        dG(j,ti) = (xi/v0*...
            ( mean(baseline( jc-d2points+1:jc+d2points+1, ic )) ...
            - mean(baseline( jc-d2points:jc+d2points, ic ))...
            )/0.001); % scaled sensitivity (y dG/dy)
    end
% figure
% plot([0:step:0.08],dG(:,ti),'-r');
end

save 'Sy_vs_y_normal.txt' dG -ascii -tabs
```

```
% Compute and store sensitivities
% Mike Myers
% 5 Jan 2012

% clear all
close all
clc

xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)

% Define step size (delta x and delta y) to evaluate temperature
step = 0.001;
xmin = -0.05;
xmax = 0.05;
n = (xmax - xmin)/step + 1; % columns
ymin = -0.15;
ymax = 0.15;
m = (ymax - ymin)/step + 1; % rows
z = 0.002667;

x = [xmin:step:xmax];
y = [ymin:step:ymax];

% simulation time t for evaluation (s)
t = [[90:1:400],[405:5:1000]];

% Import COMSOL data
baseline = [importdata(['q997_hs129_k146_x-5to5_y-15to15_z002667.txt'])];
dQt = [importdata(['q997_hs129_k146+_x-5to5_y-15to15_z002667.txt'])];

data_sets = size(baseline,1)/m; % number of timesteps in the data

% Compute time of flight
step = 0.001;

d = 0.08; % distance between sensors (m)
dpoints = d/step;
d2points = dpoints/2;

e = 0.0; % distance outside sensor grid to evaluate sensitivity (m)
epoints = 0;

plate_center_x = (n-1)/2 + 1; % column number in the data for plate center

% start_row = 10 * data_m;
% dG = zeros(m+20,n*2-1);
dG = zeros(81,1);

ic =  plate_center_x;

% t_step = [120, 150];

for ti = 1:length(t)
    [min_difference, array_position] = min(abs(t - t(ti))); ...
    plate_center_y = (array_position-1) * m + (m-1)/2 + 1;
    % row number in the data for plate center

    for j = 1:111
        % Array center is at (0cm,-7cm).
        % start array center at 0cm and go to -11cm
        % step starts -7cm below the array center and moves to the top of
        % the array 4cm
        %(???-11cm and go to 0cm
        % step starts at top of the array (4 cm) and moves to the bottom (-7cm)
        jc = plate_center_y + 1 - j;
%         jc = j - 1 + plate_center_y-(11*10);
        dG(j,ti) = (xi/v0*...
```

```
            ( mean(dQt( jc - d2points:jc + d2points, ic-d2points )) ...
            + mean(dQt( jc - d2points, ic-d2points:ic + d2points )) ...
            + mean(dQt( jc - d2points:jc + d2points, ic + d2points ))...
            + mean(dQt( jc + d2points, ic-d2points:ic + d2points ))...
            - mean(baseline( jc-d2points:jc+d2points, ic-d2points ))...
            - mean(baseline( jc - d2points, ic-d2points:ic + d2points )) ...
            - mean(baseline( jc - d2points:jc + d2points, ic + d2points ))...
            - mean(baseline( jc + d2points, ic-d2points:ic +d2points ))...
            )/0.001); % scaled sensitivity (q dG/dq)
%         dG(j,ti) = xi/v0*(mean(dQt(jc - d2points:jc + d2points, ic-d2points) -
mean(baseline( jc-d2points:jc+d2points, ic-d2points)))/0.001);
    end
% figure
% plot([0:step:0.08],dG(:,ti),'-r');
end

save 'Sk_vs_y.txt' dG -ascii -tabs

dG = [];
for ti = 1:length(t)
    [min_difference, array_position] = min(abs(t - t(ti))); ...
    plate_center_y = (array_position-1) * m + (m-1)/2 + 1; % row number in the data for plate center

    j = [-109,-89,-69,-49,-29]; % array center is at (0cm,-7cm).
    % Get data for array centers at y= -11cm, -9cm, -7cm, -5cm, and -3cm
    % Step is at the top of the array (4cm) and moves down to the bottom
    % (-4 cm)
    for ji = 1:length(j)
        jc = j(ji) - 1 + plate_center_y;
        dG(ti,ji) = (xi/v0*...
            ( mean(dQt( jc - d2points:jc + d2points, ic-d2points )) ...
            + mean(dQt( jc - d2points, ic-d2points:ic + d2points )) ...
            + mean(dQt( jc - d2points:jc + d2points, ic + d2points ))...
            + mean(dQt( jc + d2points, ic-d2points:ic + d2points ))...
            - mean(baseline( jc-d2points:jc+d2points, ic-d2points ))...
            - mean(baseline( jc - d2points, ic-d2points:ic + d2points )) ...
            - mean(baseline( jc - d2points:jc + d2points, ic + d2points ))...
            - mean(baseline( jc + d2points, ic-d2points:ic +d2points ))...
            )/0.001); % scaled sensitivity (q dG/dq)
%         dG(ti,ji) = xi/v0*(mean(dQt(jc - d2points:jc + d2points, ic-d2points) -
mean(baseline( jc-d2points:jc+d2points, ic-d2points)))/0.001);
        dG(ti,ji) = xi/v0*(mean(dQt(jc - d2points, ic-d2points:ic + d2points ) -
         mean(baseline(jc - d2points, ic-d2points:ic + d2points )))/0.001);
        G1(ti,ji) = xi/v0*mean(dQt( jc - d2points:jc + d2points, ic-d2points ));
        G2(ti,ji) = xi/v0*mean(baseline( jc - d2points:jc + d2points, ic-d2points ));
    end
end

% save 'Sk_vs_t.txt' dG -ascii -tabs

figure
plot(t,dG(:,1),'-r');
hold on
plot(t,dG(:,2),'-b');
plot(t,dG(:,3),'-g');
plot(t,dG(:,4),'-k');
plot(t,dG(:,5),'-m');
legend('-4 cm','-2 cm','0 cm', '2 cm', '4 cm')
% legend is for step location relative to the sensor grid center
% 8cm sensor spacing - 0 cm is center of grid

figure
plot(t,G1(:,1),'-r');
hold on
plot(t,G2(:,1),'-b');
legend('baseline','delta')
```

```
%{
Extended Kalman Filter for heat source parameter estimation
Mike Myers
13 January 2012

Ultrasonic pulse time of flight measurement model

Normalize change in TOF - dG/G

%}

close all
clc
x = [];
y = [];
z = [];

% noise
sensor_noise = [6e-4 6e-4 6e-4 6e-4 6e-4 6e-4];% [6e-4  6e-3 6e-2];% sec
position_noise = 0.001; % m
measurement_variance = 4e-7; % sec^2
state_variance = 0.01; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 4; % number of sensor pairs
x(1,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(1,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
z(1,:) = linspace(0.002667/2, 0.002667/2, n);

x(2,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(2,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
z(2,:) = linspace(0.002667/2, 0.002667/2, n);

x(3,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
y(3,:) = linspace(d+position_noise*randn(1),-d+position_noise*randn(1),n);
z(3,:) = linspace(0.002667/2, 0.002667/2, n);

x(4,:) = linspace(d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(4,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
z(4,:) = linspace(0.002667/2, 0.002667/2, n);

% x(5,:) = linspace(-d, d, n);
% y(5,:) = linspace(-d, d, n);
% z(5,:) = linspace(0.00635, 0.00635, n);
%
% x(6,:) = linspace(-d, d, n);
% y(6,:) = linspace(d, -d, n);
% z(6,:) = linspace(0.00635, 0.00635, n);

% Define actual step location (m)
% Step in model is at -0.07
% ys is distance from center of sensor array to bottom step
ys = [-0.06 -0.04 -0.02 0 0.02 0.035]; %[0.02 0.02 0.02];%
```

```
% Define time steps to locate source (s)
t=[100:1:130];

% Define the maximum source position
ymax = 0.15;

% Initialize saved data
dat2 = zeros(length(t)*length(ys),3);

% Define state transition matrices
g = [1];
G = eye(1); % state Jacobian
Q = state_variance * eye(1); % covariance matrix

% Define measurement model matrices
R = measurement_variance * eye(n_sensors); % covariance matrix

for j=1:length(ys)

    % Initialize EKF parameters
    mu0 = [0.0];
    Sigma0 = 0;
    Mu = mu0;
    Mubar = [];
    mutminus1 = mu0;
    Sigmatminus1 = Sigma0;
    h=[];
    Ht=[];
    Kt=[];
    Zt=[];
    walltime = [];

    fprintf('  t      mut_y  \n');

    R_ij = 2*d; % distance between sensors (m)
    dy = 0.0001;

    for i = 1:length(t)
        tic
        % Kalman filter prediction
        mubart = g * mutminus1;
        Sigmabart = G * Sigmatminus1 * G' + Q;

        % Compute measurement function (h) and measurement Jacobian (H)
        h = [];
        Ht = [];
        for k = 1:n_sensors
            %         theta(j,1) = mean(postinterp(fem, 'T',[x(j,:)-mubart(1) ;
            % y(j,:)-mubart(2) ; z(j,:) ], 'T', t(i)))-T0;
            h(k,1) = xi * (mean(postinterp(fem, 'T',[x(k,:); y(k,:)-0.07-mubart(1) ; ...
             z(k,:) ], 'T', t(i)))-T0);
            Ht(k,1) = -((xi * (mean(postinterp(fem, 'T',[x(k,:) ; y(k,:)+dy-0.07-mubart(1) ; ...
             z(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dy;

            % Get a measurement update using the simulated source
            Zt(k,1) =  xi * (mean(postinterp(fem, 'T',[x(k,:) ; y(k,:)-0.07-ys(j) ; z(k,:) ], 'T', t(i)))-T0);
        end

        Zt = Zt .* (1 + sensor_noise(j) .* randn(n_sensors,1)); % add noise to TOF measurements

        % Compute Kalman gain
        Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + R );

        % Kalman filter correction
        mut = mubart + Kt * (Zt - h);
        Sigmat = (eye(1) - Kt * Ht) * Sigmabart;
```

```
        % Limit estimated source location to points on the plate
        if abs(mut(1)) > ymax
            mut(1) = ymax*sign(mut(1));
        end

        Mu = [ Mu mut];
        Mubar = [Mubar mubart];
        Sigmatminus1 = Sigmat;
        mutminus1 = mut;

        if abs(mut(1)-ys(j)) < 0.001
            converged = [];
        else
            converged = 'unconverged';
        end
        fprintf(['%3.0f      %3.6f ',converged,' \n'],t(i), mut(1))
        dat2((j-1)*length(t)+i,:) = [ys(j) t(i) mut(1)];

        walltime(i) = toc;

    end

    fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

    % convergence_plot(Mu)

    iteration = linspace(0,length(Mu)-1,length(Mu));

    figure
    plot(t, Mu(1,1:length(t))*100,'-r','Linewidth',3)
    grid on
    xlabel('Time (s)','FontSize',12)
    ylabel('y location (cm)','FontSize',12)
    axis tight
    ylim([-12 12])

end

save 'OneWay_0.out' dat2 -ascii -tabs
```

```
%{
Simultaneous Localization and Parameter identification (SLAP)
Extended Kalman Filter for heat source localization
Mike Myers
26 January 2012

One-way ultrasonic pulse time of flight measurement model
using simulated ultrasonic data

Step source plate and model
8 cm square sensor array with 6 paths

Normalize TOF - G/G0

%}

clc
% clear all
close all

flclear xfem

% noise
sensor_noise = 6e-4; % non-dimensional
position_noise = 0.001; % m
measurement_variance = 4e-6; %4e-7
state_variance = 0.1; % m^2

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit = 297.5; %[K]

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 6; % number of sensor pairs
x(1,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(1,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
z(1,:) = linspace(0.002667/2, 0.002667/2, n);

x(2,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(2,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
z(2,:) = linspace(0.002667/2, 0.002667/2, n);

x(3,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
y(3,:) = linspace(d+position_noise*randn(1),-d+position_noise*randn(1),n);
z(3,:) = linspace(0.002667/2, 0.002667/2, n);

x(4,:) = linspace(d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(4,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
z(4,:) = linspace(0.002667/2, 0.002667/2, n);

x(5,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(5,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
z(5,:) = linspace(0.002667/2, 0.002667/2, n);

x(6,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(6,:) = linspace(d+position_noise*randn(1), -d+position_noise*randn(1), n);
```

```
z(6,:) = linspace(0.002667/2, 0.002667/2, n);

% Sensor positions for the model (no position noise)
xm(1,:) = linspace(-d, -d, n);
ym(1,:) = linspace(-d, d, n);
zm(1,:) = linspace(0.002667/2, 0.002667/2, n);

xm(2,:) = linspace(-d, d, n);
ym(2,:) = linspace(d, d, n);
zm(2,:) = linspace(0.002667/2, 0.002667/2, n);

xm(3,:) = linspace(d, d, n);
ym(3,:) = linspace(d,-d,n);
zm(3,:) = linspace(0.002667/2, 0.002667/2, n);

xm(4,:) = linspace(d, -d, n);
ym(4,:) = linspace(-d, -d, n);
zm(4,:) = linspace(0.002667/2, 0.002667/2, n);

xm(5,:) = linspace(-d, d, n);
ym(5,:) = linspace(-d, d, n);
zm(5,:) = linspace(0.002667/2, 0.002667/2, n);

xm(6,:) = linspace(-d, d, n);
ym(6,:) = linspace(d, -d, n);
zm(6,:) = linspace(0.002667/2, 0.002667/2, n);

% Define starting actual parameters
yq = 0;
q1 = 0;    % [W/m^2]
h_edges = 3;   %[W/m^2 K]
h_sides = 12.9; %[W/m^2 K]
e_polished = 0.075; % emmissitivity of polished area of plate
e_paint = 0.8; % emmissivity of painted area of plate
sigma = 5.67e-8; % [W/m^2 K^4] Stefan-Boltzmann constant
dt = 1;        % (s)

% Define time steps to locate source (s)
t=[100:1:200];

% Define the maximum source position
ymax = 0.037;
ymin = -0.06;

% Initialize saved data
dat2 = zeros(length(t),3);

% Define state transition matrices
g = [1];
G = eye(1); % state Jacobian
Q = state_variance * eye(1); % covariance matrix

% Define measurement model matrices
R = measurement_variance * eye(n_sensors); % covariance matrix

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = 'a';
vrsn.major = 0;
vrsn.build = 608;
vrsn.rcs = '$Name: v35ap $';
vrsn.date = '$Date: 2009/05/11 07:38:49 $';
xfem.version = vrsn;

% Geometry 2
g1=square2('0.305','base','center','pos',{'0','0'},'rot','0');
g2=rect2('0.19','0.14','base','center','pos',{'0','0'},'rot','0');
```

```
flclear fem

% Analyzed geometry
clear s
s.objs={g1,g2};
s.name={'SQ1','R1'};
s.tags={'g1','g2'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Initialize mesh for geometry 2
fem.mesh=meshinit(fem, ...
    'hauto',5, ...
    'hmaxsub',[2,100e-3]);
xfem.fem{2}=fem;
flclear fem

% Extrude mesh for geometry 2
[extgeom, extmesh]=meshextrude(xfem.fem{2}, ...
    'elextlayers',{[0 1]}, ...
    'distance',[0.002667], ...
    'scale',[1;1], ...
    'displ',[0;0], ...
    'twist',[0], ...
    'face','none', ...
    'wrkpln',[0 1 0;0 0 1;0 0 0], ...
    'out',{'geom','mesh'});
fem.geom=extgeom;
fem.mesh=extmesh;

% Constants
xfem.const = {'Tamb',Tinit, ...
    'T0',Tinit, ...
    'Tinf',Tinit, ...
    'Tr',Tinit, ...
    'q_in',q1, ...
    'h_sides',h_sides, ...
    'h_edges',h_edges, ...
    'Const_polished',e_polished * sigma, ...
    'Const_paint',e_paint * sigma};


xfem.fem{1}=fem;

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'HeatTransfer';
appl.sshape = 2;
appl.assignsuffix = '_ht';
clear bnd
bnd.Tamb = {'Tamb',0,'Tamb','Tamb'};
bnd.name = {'Heated Zone','','Edges','Sides'};
bnd.Const = {'Const_paint',0,'Const_polished','Const_polished'};
bnd.q0 = {'q_in',0,0,0};
bnd.type = {'q','cont','q','q'};
bnd.Tinf = {'Tinf',273.15,'Tinf','Tinf'};
bnd.h = {'h_sides',0,'h_edges','h_sides'};
bnd.ind = [3,3,4,4,3,2,2,1,4,2,2,3];
appl.bnd = bnd;
clear equ
equ.C = 500;
equ.init = 'T0';
equ.k = 14.6;
equ.rho = 8000;
equ.ind = [1,1];
```

```
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
fem.outform = 'general';
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{1} = fem;

fem=xfem.fem{2};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{2} = fem;

% ODE Settings
clear ode
clear units;
units.basesystem = 'SI';
ode.units = units;
xfem.ode=ode;

% Multiphysics
xfem=multiphysics(xfem);

% Extend mesh
xfem.xmesh=meshextend(xfem);

% Solve problem
xfem.sol=femtime(xfem, ...
    'solcomp',{'T'}, ...
    'outcomp',{'T'}, ...
    'blocksize','auto', ...
    'tlist',1, ...
    'tout','tlist', ...
    'tsteps','strict', ...
    'linsolver','cg', ...
    'prefun','amg');

% Save current fem structure for restart purposes
fem0=xfem;

% Store solution for actual state (simulated)
fem_actual = fem0;
fem_actual.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

% Store solution for estimated state
fem_y = fem0;
fem_y.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

fprintf('  t      mut_y    Sigmat\n');

R_ij = 2*d; % distance between sensors (m)
dy = 0.0001;


% Define actual state
yq_limit = 0.03; %[m]

% Sawtooth pattern
yqv = 0;%0.002; % velocity [m/2]
yq(1) = -0.02;
for i=2:length(t)
    if abs(yq(i-1))>=yq_limit
        yqv = -yqv;
    end
```

```
    yq(i) = yq(i-1) + yqv;
end

% Sine wave pattern
% yq = sin((t-t(1))./60*2*pi) * yq_limit + yq(1);

q1 = 9.97e3;    % [W/m^2]

% Define source location velocity (m/s)

% Define starting guess (this really goes into mut - redo this to add all parameters to the state!!!!!)
yq_e = 0;%0.02;            % [m]
q1_e = q1;         % [W/m^2]
h_edges_e = h_edges;   %[W/m^2 K]
h_sides_e = h_sides;   %[W/m^2 K]

mu0 = [yq_e];
% Initialize EKF parameters
Sigma0 = 0;
Mu = mu0;
Mubar = [];
mutminus1 = mu0;
Sigmatminus1 = Sigma0;
h=[];
Ht=[];
Kt=[];
Zt=[];
walltime = [];

Sigmat_history = [];
Kt_history = [];
Ht_history = [];
h_history = [];
Zt_history = [];
yq_history = [yq];

dat2(1,:) = [yq(1) t(1) yq_e];

for i=2:length(t)
    tic
    % Kalman filter prediction
    mubart = g * mutminus1;
    Sigmabart = G * Sigmatminus1 * G' + Q;

    % Compute measurement function (h) and measurement Jacobian (H)
    h = [];
    Ht = [];

    % Get solution for actual source (simulated)
    [xfem temp] = Step_heat(yq(i),q1,h_edges,h_sides,e_polished* ...
     sigma,e_paint*sigma,Tinit,dt,0,fem_actual);

    % Save current fem structure for restart purposes
    fem0=xfem;

    % Store solution for actual state (simulated)
    fem_actual = fem0;
    fem_actual.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

    % Get solution for estimated state
    [xfem temp] = Step_heat(mubart(1),q1_e,h_edges_e,h_sides_e,e_polished* ...
     sigma,e_paint*sigma,Tinit,dt,0,fem_y);
    fem0=xfem;
    fem_y = fem0;
    fem_y.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

    for k = 1:n_sensors
        h(k,1) = xi * (mean(postinterp(fem_y, 'T',[xm(k,:); ym(k,:)-0.07 ;  ...
```

```
     zm(k,:) ], 'T', t(i)))-T0);
    Ht(k,1) = ((xi * (mean(postinterp(fem_y, 'T',[xm(k,:) ; ym(k,:)-0.07+dy ; ...
     zm(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / -dy;
    %         Ht(k,1)=0;
    % Get a measurement update using the simulated source
    Zt(k,1) =  xi * (mean(postinterp(fem_actual, 'T',[x(k,:) ; y(k,:)-0.07 ; z(k,:) ], 'T', t(i)))-T0);
end

Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

% Compute Kalman gain
%     Kt = Sigmabart * Ht' * inv( Ht * Sigmabart * Ht' + R )
Kt = Sigmabart * Ht' / ( Ht * Sigmabart * Ht' + R );

% Kalman filter correction
mut = mubart + Kt * (Zt - h);
Sigmat = (eye(1) - Kt * Ht) * Sigmabart;

% Limit estimated source location to points on the plate
if mut(1) > ymax
    mut(1) = ymax;
else if mut(1) < ymin
        mut(1) = ymin;
    end
end

%   mut = mutminus1;

Mu = [ Mu mut];
Mubar = [Mubar mubart];
Sigmatminus1 = Sigmat;
mutminus1 = mut;


if abs(mut(1)-yq) < 0.001
    converged = [];
else
    converged = 'unconverged';
end
fprintf(['%3.0f      %3.6f     %3.6f    ',converged,' \n'],t(i), mut(1), Sigmat)
dat2(i,:) = [yq(i) t(i) mut(1)];

walltime(i) = toc;
h_history = [h_history
    h'];
Ht_history = [Ht_history
    Ht'];
Sigmat_history = [Sigmat_history Sigmat];
Kt_history = [Kt_history
    Kt];
Zt_history = [Zt_history
    Zt'];
%     yq_history = [yq_history
%         yq(i)];
%     yq = yq + yqv;
%     Zt - h
%    pause

figure %('position',[50,50,1000,500])
postplot(fem_y, ...
    'tetdata',{'T','cont','internal','unit','K'}, ...
    'tetmap','Rainbow', ...
    'tetkeep',1, ...
    'tetkeeptype','random', ...
    'solnum','end', ...
    'title',['Time=', num2str(t(i)), '    Subdomain: Temperature [K]'], ...
    'geom','off', ...
    'grid','on', ...
```

```
        'campos',[-1.137027353993085,-1.4818028973382882,1.0796919794268787], ...
        'camtarget',[0,0,0.00133350002579391], ...
        'camup',[0,0,1], ...
        'camva',6.4527092929162855);

    print('-djpeg', ['Movie\test', num2str(t(i)),'.jpg'])
    stop
end

fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

% Plot convergence data

iteration = linspace(0,length(Mu)-1,length(Mu));

figure
plot(t, yq(1:length(t))*100,'-b','Linewidth',3)
hold on
plot(t, Mu(1,1:length(t))*100,'-r','Linewidth',3)
grid on
xlabel('Time (s)','FontSize',12)
ylabel('y location (cm)','FontSize',12)
axis tight
ylim([-12 12])

% Plot solution for entire plate
figure
postplot(fem_actual, ...
    'tetdata',{'T','cont','internal','unit','K'}, ...
    'tetmap','Rainbow', ...
    'tetkeep',1, ...
    'tetkeeptype','random', ...
    'solnum','end', ...
    'title','Time=1    Subdomain: Temperature [K]', ...
    'geom','off', ...
    'grid','on', ...
    'campos',[-1.137027353993085,-1.4818028973382882,1.0796919794268787], ...
    'camtarget',[0,0,0.00133350002579391], ...
    'camup',[0,0,1], ...
    'camva',6.4527092929162855);

figure
postplot(fem_y, ...
    'tetdata',{'T','cont','internal','unit','K'}, ...
    'tetmap','Rainbow', ...
    'tetkeep',1, ...
    'tetkeeptype','random', ...
    'solnum','end', ...
    'title','Time=1    Subdomain: Temperature [K]', ...
    'geom','off', ...
    'grid','on', ...
    'campos',[-1.137027353993085,-1.4818028973382882,1.0796919794268787], ...
    'camtarget',[0,0,0.00133350002579391], ...
    'camup',[0,0,1], ...
    'camva',6.4527092929162855);

tmaxminus1 = length(t)-1;

figure
plot(t(1:tmaxminus1), Sigmat_history)
title('\Sigma')

figure
plot(t(1:tmaxminus1), Kt_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Kt_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Kt_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Kt_history(:,4),'-k','Linewidth',3)
```

```
title('Kt')

figure
plot(t(1:tmaxminus1), h_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), h_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), h_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), h_history(:,4),'-k','Linewidth',3)
title('h')

figure
plot(t(1:tmaxminus1), Zt_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Zt_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Zt_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Zt_history(:,4),'-k','Linewidth',3)
title('Zt')

figure
plot(t(1:tmaxminus1), Ht_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Ht_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Ht_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Ht_history(:,4),'-k','Linewidth',3)
title('Ht')

% Save convergence data to disk
save 'Plate_moving_step_yn2.out' dat2 -ascii -tabs
```

```
%{
Simultaneous Localization and Parameter identification (SLAP)
Extended Kalman Filter for heat source localization
Mike Myers
26 January 2012

One-way ultrasonic pulse time of flight measurement model
using simulated ultrasonic data

Step source plate and model
8 cm square sensor array with 6 paths

Normalize TOF - G/G0

%}

clc
% clear all
close all

flclear xfem

% noise
sensor_noise = 6e-4; % non-dimensional
position_noise = 0.001; % m
measurement_variance = 4e-6; %4e-7
state_variance = 1e7; % W^2/m^4

% Initialize random number generator
randn('state',sum(100*clock))

% Define material properties
xi = 110e-6; % Time of flight temperature factor (1/K)
v0 = 5100; % reference sound speed (m/s)
T0 = 293; % temperature for reference sound speed (K)
Tinit = 297.5; %[K]

% Define sensor locations (m)
%
d = 0.04; % sensor location distance from origin
n = 100; % number of points between sensors to retrieve temperature
(for computing average temp between sensors)
x = [];
y = [];
z = [];
n_sensors = 6; % number of sensor pairs
x(1,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(1,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
z(1,:) = linspace(0.002667/2, 0.002667/2, n);

x(2,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(2,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
z(2,:) = linspace(0.002667/2, 0.002667/2, n);

x(3,:) = linspace(d+position_noise*randn(1), d+position_noise*randn(1), n);
y(3,:) = linspace(d+position_noise*randn(1),-d+position_noise*randn(1),n);
z(3,:) = linspace(0.002667/2, 0.002667/2, n);

x(4,:) = linspace(d+position_noise*randn(1), -d+position_noise*randn(1), n);
y(4,:) = linspace(-d+position_noise*randn(1), -d+position_noise*randn(1), n);
z(4,:) = linspace(0.002667/2, 0.002667/2, n);

x(5,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(5,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
z(5,:) = linspace(0.002667/2, 0.002667/2, n);

x(6,:) = linspace(-d+position_noise*randn(1), d+position_noise*randn(1), n);
y(6,:) = linspace(d+position_noise*randn(1), -d+position_noise*randn(1), n);
```

```
z(6,:) = linspace(0.002667/2, 0.002667/2, n);

% Sensor positions for the model (no position noise)
xm(1,:) = linspace(-d, -d, n);
ym(1,:) = linspace(-d, d, n);
zm(1,:) = linspace(0.002667/2, 0.002667/2, n);

xm(2,:) = linspace(-d, d, n);
ym(2,:) = linspace(d, d, n);
zm(2,:) = linspace(0.002667/2, 0.002667/2, n);

xm(3,:) = linspace(d, d, n);
ym(3,:) = linspace(d,-d,n);
zm(3,:) = linspace(0.002667/2, 0.002667/2, n);

xm(4,:) = linspace(d, -d, n);
ym(4,:) = linspace(-d, -d, n);
zm(4,:) = linspace(0.002667/2, 0.002667/2, n);

xm(5,:) = linspace(-d, d, n);
ym(5,:) = linspace(-d, d, n);
zm(5,:) = linspace(0.002667/2, 0.002667/2, n);

xm(6,:) = linspace(-d, d, n);
ym(6,:) = linspace(d, -d, n);
zm(6,:) = linspace(0.002667/2, 0.002667/2, n);

% Define starting actual parameters
yq = 0;
q1 = 0;    % [W/m^2]
h_edges = 3;   %[W/m^2 K]
h_sides = 12.9; %[W/m^2 K]
e_polished = 0.075; % emmissitivity of polished area of plate
e_paint = 0.8; % emmissivity of painted area of plate
sigma = 5.67e-8; % [W/m^2 K^4] Stefan-Boltzmann constant
dt = 1;        % (s)

% Define time steps to locate source (s)
t=[100:dt:100+dt*500];

% Define the maximum source heat flux
qmax = 15e3; % [W/m^2]
qmin = 0;

% Initialize saved data
dat2 = zeros(length(t),3);

% Define state transition matrices
g = [1];
G = eye(1); % state Jacobian
Q = state_variance * eye(1); % covariance matrix

% Define measurement model matrices
R = measurement_variance * eye(n_sensors); % covariance matrix

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = 'a';
vrsn.major = 0;
vrsn.build = 608;
vrsn.rcs = '$Name: v35ap $';
vrsn.date = '$Date: 2009/05/11 07:38:49 $';
xfem.version = vrsn;

% Geometry 2
g1=square2('0.305','base','center','pos',{'0','0'},'rot','0');
g2=rect2('0.19','0.14','base','center','pos',{'0','0'},'rot','0');
```

```
flclear fem

% Analyzed geometry
clear s
s.objs={g1,g2};
s.name={'SQ1','R1'};
s.tags={'g1','g2'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Initialize mesh for geometry 2
fem.mesh=meshinit(fem, ...
    'hauto',5, ...
    'hmaxsub',[2,100e-3]);
xfem.fem{2}=fem;
flclear fem

% Extrude mesh for geometry 2
[extgeom, extmesh]=meshextrude(xfem.fem{2}, ...
    'elextlayers',{[0 1]}, ...
    'distance',[0.002667], ...
    'scale',[1;1], ...
    'displ',[0;0], ...
    'twist',[0], ...
    'face','none', ...
    'wrkpln',[0 1 0;0 0 1;0 0 0], ...
    'out',{'geom','mesh'});
fem.geom=extgeom;
fem.mesh=extmesh;

% Constants
xfem.const = {'Tamb',Tinit, ...
    'T0',Tinit, ...
    'Tinf',Tinit, ...
    'Tr',Tinit, ...
    'q_in',q1, ...
    'h_sides',h_sides, ...
    'h_edges',h_edges, ...
    'Const_polished',e_polished * sigma, ...
    'Const_paint',e_paint * sigma};


xfem.fem{1}=fem;

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'HeatTransfer';
appl.sshape = 2;
appl.assignsuffix = '_ht';
clear bnd
bnd.Tamb = {'Tamb',0,'Tamb','Tamb'};
bnd.name = {'Heated Zone','','Edges','Sides'};
bnd.Const = {'Const_paint',0,'Const_polished','Const_polished'};
bnd.q0 = {'q_in',0,0,0};
bnd.type = {'q','cont','q','q'};
bnd.Tinf = {'Tinf',273.15,'Tinf','Tinf'};
bnd.h = {'h_sides',0,'h_edges','h_sides'};
bnd.ind = [3,3,4,4,3,2,2,1,4,2,2,3];
appl.bnd = bnd;
clear equ
equ.C = 500;
equ.init = 'T0';
equ.k = 14.6;
equ.rho = 8000;
equ.ind = [1,1];
```

```
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
fem.outform = 'general';
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{1} = fem;

fem=xfem.fem{2};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{2} = fem;

% ODE Settings
clear ode
clear units;
units.basesystem = 'SI';
ode.units = units;
xfem.ode=ode;

% Multiphysics
xfem=multiphysics(xfem);

% Extend mesh
xfem.xmesh=meshextend(xfem);

% Solve problem
xfem.sol=femtime(xfem, ...
    'solcomp',{'T'}, ...
    'outcomp',{'T'}, ...
    'blocksize','auto', ...
    'tlist',1, ...
    'tout','tlist', ...
    'tsteps','strict', ...
    'linsolver','cg', ...
    'prefun','amg');

% Save current fem structure for restart purposes
fem0=xfem;

% Store solution for actual state (simulated)
fem_actual = fem0;
fem_actual.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

% Store solution for estimated state
fem_baseline = fem0;
fem_baseline.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

fprintf('  t      mut_q    Sigmat\n');

R_ij = 2*d; % distance between sensors (m)
dy = 0.0001;
dq = 10;

% Define actual state
yq = -0.04;      % [m]
q1(1) = 9.97e3;    % [W/m^2]
q_limit = 1e3; %[m]

% Sine wave pattern
q1 = sin((t-t(1))./60*2*pi) * q_limit + q1(1);

% Define source location velocity (m/s)
yqv = 0;
```

```
% Define starting guess (this really goes into mut - redo this to add all parameters to the state!!!!!)
yq_e = -0.04;            % [m]
q1_e = 9.97e3;%9.97e3;         % [W/m^2]
h_edges_e = h_edges;   %[W/m^2 K]
h_sides_e = h_sides;   %[W/m^2 K]

mu0 = [q1_e];
% Initialize EKF parameters
Sigma0 = 0;
Mu = mu0;
Mubar = [];
mutminus1 = mu0;
Sigmatminus1 = Sigma0;
h=[];
Ht=[];
Kt=[];
Zt=[];
walltime = [];

Sigmat_history = [];
Kt_history = [];
Ht_history = [];
h_history = [];
Zt_history = [];

dat2(1,:) = [q1(1) t(1) q1_e];

for i=2:length(t)
    tic
    % Kalman filter prediction
    mubart = g * mutminus1;
    Sigmabart = G * Sigmatminus1 * G' + Q;

    % Compute measurement function (h) and measurement Jacobian (H)
    h = [];
    Ht = [];

    % Get solution for actual source (simulated)
    [xfem temp] = Step_heat(yq,q1(i),h_edges,h_sides,e_polished* ...
     sigma,e_paint*sigma,Tinit,dt,0,fem_actual);

    % Save current fem structure for restart purposes
    fem0=xfem;

    % Store solution for actual state (simulated)
    fem_actual = fem0;
    fem_actual.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

    % Get solutions for estimated state
    [xfem xfemdq] = Step_heat(yq_e,mubart(1),h_edges_e,h_sides_e,e_polished* ...
     sigma,e_paint*sigma,Tinit,dt,dq,fem_baseline);
    fem0=xfem;
    fem_baseline = fem0;
    fem_baseline.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);
    fem0=xfemdq;
    fem_dq = fem0;
    fem_dq.sol = asseminit(fem0,'init',fem0.sol,'solnum',[2]);

    for k = 1:n_sensors
        h(k,1) = xi * (mean(postinterp(fem_baseline, 'T',[xm(k,:); ym(k,:)-0.07 ;  ...
         zm(k,:) ], 'T', t(i)))-T0);
        Ht(k,1) = ((xi * (mean(postinterp(fem_dq, 'T',[xm(k,:) ; ym(k,:)-0.07 ; ...
         zm(k,:) ], 'T', t(i)))-T0)) - h(k,1)) / dq;
        % Get a measurement update using the simulated source
        Zt(k,1) =  xi * (mean(postinterp(fem_actual, 'T',[x(k,:) ; y(k,:)-0.07 ; ...
         z(k,:) ], 'T', t(i)))-T0);
    end
```

```
    Zt = Zt .* (1 + sensor_noise .* randn(n_sensors,1)); % add noise to TOF measurements

    % Compute Kalman gain
    Kt = Sigmabart * Ht' / ( Ht * Sigmabart * Ht' + R );

    % Kalman filter correction
    mut = mubart + Kt * (Zt - h);
    Sigmat = (eye(1) - Kt * Ht) * Sigmabart;

    % Limit estimated source location to points on the plate
    %     if mut(1) > qmax
    %         mut(1) = qmax;
    %     else if mut(1) < qmin
    %             mut(1) = qmin;
    %         end
    %     end

    %   mut = mutminus1;

    Mu = [ Mu mut];
    Mubar = [Mubar mubart];
    Sigmatminus1 = Sigmat;
    mutminus1 = mut;


    if abs(mut(1)-q1) < 500
        converged = [];
    else
        converged = 'unconverged';
    end
    fprintf(['%3.0f      %3.6f     %3.6f    ',converged,' \n'],t(i), mut(1), Sigmat)
    dat2(i,:) = [q1(i) t(i) mut(1)];

    walltime(i) = toc;
    h_history = [h_history
        h'];
    Ht_history = [Ht_history
        Ht'];
    Sigmat_history = [Sigmat_history Sigmat];
    Kt_history = [Kt_history
        Kt];
    Zt_history = [Zt_history
        Zt'];

    %     Zt - h
    %    pause
end

fprintf('\n Average wall time per iteration = %2.3f sec\n\n', mean(walltime));

% Plot convergence data

iteration = linspace(0,length(Mu)-1,length(Mu));

figure
plot(t, Mu(1,1:length(t)),'-r','Linewidth',3)
grid on
xlabel('Time (s)','FontSize',12)
ylabel('Heat flux (W/m^2)','FontSize',12)
axis tight
% ylim([-12 12])


% Plot solution for entire plate
figure
postplot(fem_actual, ...
    'tetdata',{'T','cont','internal','unit','K'}, ...
```

```
    'tetmap','Rainbow', ...
    'tetkeep',1, ...
    'tetkeeptype','random', ...
    'solnum','end', ...
    'title','Time=1    Subdomain: Temperature [K]', ...
    'grid','on', ...
    'campos',[-1.137027353993085,-1.4818028973382882,1.0796919794268787], ...
    'camtarget',[0,0,0.00133350002579391], ...
    'camup',[0,0,1], ...
    'camva',6.4527092929162855);

figure
postplot(fem_baseline, ...
    'tetdata',{'T','cont','internal','unit','K'}, ...
    'tetmap','Rainbow', ...
    'tetkeep',1, ...
    'tetkeeptype','random', ...
    'solnum','end', ...
    'title','Time=1    Subdomain: Temperature [K]', ...
    'grid','on', ...
    'campos',[-1.137027353993085,-1.4818028973382882,1.0796919794268787], ...
    'camtarget',[0,0,0.00133350002579391], ...
    'camup',[0,0,1], ...
    'camva',6.4527092929162855);

tmaxminus1 = length(t)-1;
figure
plot(t(1:tmaxminus1), Sigmat_history)
title('\Sigma')

figure
plot(t(1:tmaxminus1), Kt_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Kt_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Kt_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Kt_history(:,4),'-k','Linewidth',3)
title('Kt')

figure
plot(t(1:tmaxminus1), h_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), h_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), h_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), h_history(:,4),'-k','Linewidth',3)
title('h')

figure
plot(t(1:tmaxminus1), Zt_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Zt_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Zt_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Zt_history(:,4),'-k','Linewidth',3)
title('Zt')

figure
plot(t(1:tmaxminus1), Ht_history(:,1),'-r','Linewidth',3)
hold on
plot(t(1:tmaxminus1), Ht_history(:,2),'-b','Linewidth',3)
plot(t(1:tmaxminus1), Ht_history(:,3),'-g','Linewidth',3)
plot(t(1:tmaxminus1), Ht_history(:,4),'-k','Linewidth',3)
title('Ht')

% Save convergence data to disk
save 'Plate_moving_step_q_dt1.out' dat2 -ascii -tabs
```

```
function [xfem_baseline xfem_dq] = Step_heat(yq, q1, h_edges, h_sides,
Const_polished, Const_paint, Tinit, dt, dq, fem1)

% Geometry 2
g1=square2('0.305','base','center','pos',{'0','0'},'rot','0');
g2=rect2('0.19','0.14','base','center','pos',{'0',yq},'rot','0');
flclear fem

% Analyzed geometry
clear s
s.objs={g1,g2};
s.name={'SQ1','R1'};
s.tags={'g1','g2'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Initialize mesh for geometry 2
fem.mesh=meshinit(fem, ...
    'hauto',5, ...
    'hmaxsub',[2,100e-3]);
xfem.fem{2}=fem;
flclear fem

% Extrude mesh for geometry 2
[extgeom, extmesh]=meshextrude(xfem.fem{2}, ...
    'elextlayers',{[0 1]}, ...
    'distance',[0.002667], ...
    'scale',[1;1], ...
    'displ',[0;0], ...
    'twist',[0], ...
    'face','none', ...
    'wrkpln',[0 1 0;0 0 1;0 0 0], ...
    'out',{'geom','mesh'});
fem.geom=extgeom;
fem.mesh=extmesh;

% Constants
xfem.const = {'Tamb',Tinit, ...
    'T0',Tinit, ...
    'Tinf',Tinit, ...
    'Tr',Tinit, ...
    'q_in',q1, ...
    'h_sides',h_sides, ...
    'h_edges',h_edges, ...
    'Const_polished',Const_polished, ...
    'Const_paint',Const_paint};

% (Default values are not included)

xfem.fem{1}=fem;

fem=xfem.fem{1};

% Application mode 1
clear appl
appl.mode.class = 'HeatTransfer';
appl.sshape = 2;
appl.assignsuffix = '_ht';
clear bnd
bnd.Tamb = {'Tamb',0,'Tamb','Tamb'};
bnd.name = {'Heated Zone','','Edges','Sides'};
bnd.Const = {'Const_paint',0,'Const_polished','Const_polished'};
bnd.q0 = {'q_in',0,0,0};
bnd.type = {'q','cont','q','q'};
bnd.Tinf = {'Tinf',273.15,'Tinf','Tinf'};
bnd.h = {'h_sides',0,'h_edges','h_sides'};
bnd.ind = [3,3,4,4,3,2,2,1,4,2,2,3];
```

```
appl.bnd = bnd;
clear equ
equ.C = 500;
equ.init = 'T0';
equ.k = 14.6;
equ.rho = 8000;
equ.ind = [1,1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
fem.outform = 'general';
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{1} = fem;

fem=xfem.fem{2};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;
xfem.fem{2} = fem;

% ODE Settings
clear ode
clear units;
units.basesystem = 'SI';
ode.units = units;
xfem.ode=ode;

% Multiphysics
xfem=multiphysics(xfem);

% Extend mesh
xfem.xmesh=meshextend(xfem);

% Transfer previous solution to new mesh
xfem.sol = asseminit(xfem,'init',fem1);

% Solve problem
xfem.sol=femtime(xfem, ...
    'init',xfem.sol, ...
    'solcomp',{'T'}, ...
    'outcomp',{'T'}, ...
    'blocksize','auto', ...
    'tlist',dt, ...
    'tout','tlist', ...
    'tsteps','strict', ...
    'linsolver','cg', ...
    'prefun','amg');

% Store solution in baseline structure
xfem_baseline = xfem;

if dq == 0
    xfem_dq = 0;
else
    % Modify constants for dq
    xfem.const = {'q_in',q1+dq};

    % Transfer previous solution to new mesh
    xfem.sol = asseminit(xfem,'init',fem1);

    % Solve problem
    xfem.sol=femtime(xfem, ...
        'init',xfem.sol, ...
        'solcomp',{'T'}, ...
```

```
        'outcomp',{'T'}, ...
        'blocksize','auto', ...
        'tlist',dt, ...
        'tout','tlist', ...
        'tsteps','strict', ...
        'linsolver','cg', ...
        'prefun','amg');

    % Store solution in dq structure
    xfem_dq = xfem;
end

end
```

BIBLIOGRAPHY

[Arulampalam et al., 2002] Arulampalam, M., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE transactions on signal processing*, 50(2):174–188.

[Augereau et al., 2007] Augereau, F., Laux, D., Allais, L., Mottot, M., and Caes, C. (2007). Ultrasonic measurement of anisotropy and temperature dependence of elastic parameters by a dry coupling method applied to a 6061-T6 alloy. *Ultrasonics*, 46:34–41.

[Bar-Shalom and Li, 2001] Bar-Shalom, Y. and Li, X.-R. (2001). *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA.

[Beck and Woodbury, 1998] Beck, J. and Woodbury, K. (1998). Inverse problems and parameter estimation: integration of measurements and analysis. *Measurement Science and Technology*, 9:839–847.

[Beck and Arnold, 1977] Beck, J. V. and Arnold, K. J. (1977). *Parameter Estimation in Engineering and Science*. John Wiley & Sons.

[Berger et al., 2009] Berger, K., Rufer, S., Kimmel, R., and Adamczak, D. (2009). Aerothermodynamic characteristics of boundary layer transition and trip effectiveness of the HIFiRE flight 5 vehicle. In *AIAA Proceedings*, number AIAA-2009-4055.

[Bertin and Cummings, 2003] Bertin, J. and Cummings, R. (2003). Fifty years of hypersonics: where we've been, where we're going. *Progress in Aerospace Sciences*, 39(6-7):511–536.

[Bertsekas, 1996] Bertsekas, D. P. (1996). Incremental least squares methods and the extended Kalman filter. *SIAM Journal on Optimization*, 6(3):807–822.

[Bretscher, 1995] Bretscher, O. (1995). *Linear Algebra With Applications*. Prentice Hall.

[Burgers et al., 1998] Burgers, G., van Leeuwen, P., and Evensen, G. (1998). Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, 126:1719–1724.

[Doe, 2012] Doe, R. (2012). The engineering toolbox. [Online; accessed 5-Jan-2012].

[Doucet et al., 2002] Doucet, A., Gordon, N., and Krishnamurthy, V. (2002). Particle filters for state estimation of jump Markov linear systems. *IEEE Transactions on Signal Processing*, 49(3):613–624.

[Dowding and Blackwell, 2001] Dowding, K. J. and Blackwell, B. F. (2001). Sensitivity analysis for nonlinear heat conduction. *Journal of Heat Transfer*, 123(1):1.

[Fay and Riddell, 1958] Fay, J. and Riddell, F. (1958). Theory of stagnation point heat transfer in dissociated air. *Journal of the Aeronautical Sciences*, 25(2):73–85.

[Frankel et al., 2010] Frankel, J., Keyhani, M., Elkins, B., and Arimilli, R. (2010). New in situ method for estimating thermal diffusivity using rate-based temperature sensors. *Journal of Thermophysics and Heat Transfer*, 24(4):811–817.

[Gai and Hayne, 2010] Gai, S. and Hayne, M. (2010). Heat transfer behind a step in high-enthalpy laminar hypersonic flow. *Journal of Thermophysics and Heat Transfer*, 24(4):839–841.

[Gauffre, 1988] Gauffre, G. (1988). Détection de la transition laminaire turbulent par thermographie infrarouge (Detection of laminar-turbulent transition by infrared thermography). *La Recherche Aérospatiale*, (2):11–22.

[Hein, 1993] Hein (1993). Current time-domain methods for assessing tissue motion by analysis from reflected ultrasound echoes-a review. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 40(2):84–102.

[Horvath et al., 2002] Horvath, T. J., Berry, S. A., and Hollis, B. R. (2002). Boundary layer transition on slender cones in conventional and low disturbance Mach 6 wind tunnels. In *32nd AIAA Fluid Dynamics Conference and Exhibit*, number AIAA-2002-2743, St. Louis, MO. AIAA.

[Hoyle and Luke, 1994] Hoyle, B. S. and Luke, S. P. (1994). Ultrasound in the process industries. *Engineering Science and Education Journal*, 3(3):119–122.

[Incropera and DeWitt, 2002] Incropera, F. P. and DeWitt, D. P. (2002). *Introduction to Heat Transfer*. John Wiley and Sons, Hoboken, NJ, 4th edition.

[Kendall, 1975] Kendall, J. (1975). Wind tunnel experiments relating to supersonic and hypersonic boundary-layer transition. *AIAA Journal*, 13(3):290–299.

[Kimmel et al., 2007] Kimmel, R., Adamczak, D., Gaitonde, D., Rougeux, A., and Hayes, J. (2007). HIFiRE-1 boundary layer transition experiment design. In *proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit*, volume 2007-534, Reno, NV, USA.

[Konno et al., 1993] Konno, M., Cui, A., Nishiwaki, N., and Hori, S. (1993). Measurement of the polymer melt temperature in injection molding machine by using ultrasonic technique. In *proceedings of the 51st Annual Technical Conference, Society of Plastics Engineers*, pages 2798–2803, New Orleans, LA, USA.

[Kozlov et al., 1989] Kozlov, V., Adamchik, V., and Lipovtsev, V. (1989). Local heating of an unbounded orthotropic plate through a circular and annular domain. *Journal of Engineering Physics and Thermophysics*, 57:1381–1391.

[Lin, 1991] Lin, C. (1991). *Modern navigation, guidance, and control processing*. Prentice Hall series in advanced navigation, guidance, and control, and their applications. Prentice Hall.

[Liu, 1984] Liu, J. M. (1984). Temperature dependence of elastic stiffness in aluminum alloys measured with non-contact electromagnetic acoustic transducers (EMATS). In *IEEE Ultrasonics Symposium*, pages 972–974.

[Liu et al., 2010] Liu, T., Cai, Z., Lai, J., Rabal, J., and Sullivan, J. P. (2010). Analytical method for determining heat flux from temperature-sensitive-paint measurements in hypersonic tunnels. *Journal of Thermophysics and Heat Transfer*, 24(1):1–10.

[Loupas, 1995] Loupas (1995). An axial velocity estimator for ultrasound blood flow imaging, based on a full evaluation of the Doppler equation by means of a two-dimensional autocorrelation approach. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 42(4):672–688.

[Lynnworth and Papadakis, 1970] Lynnworth, L. and Papadakis, E. (1970). Ultrasonic thermometry. In *Ultrasonics Symposium*, San Francisco, CA. IEEE.

[Majji et al., 2010] Majji, M., Juang, J.-N., and Junkins, J. L. (2010). Observer/Kalman-filter time-varying system identification. *Journal of Guidance, Control, and Dynamics*, 33(3):887–900.

[Malik, 1989] Malik, M. (1989). Prediction and control of transition in supersonic and hypersonic boundary layers. *AIAA Journal*, 27(11):1487–1493.

[Marioli et al., 1992] Marioli, D., Narduzzi, C., Offelli, C., Petri, D., Sardini, E., and Taroni, A. (1992). Digital time-of-flight measurement for ultrasonic sensors. *IEEE Transactions on Instrumentation and Measurement*, 41(1):93–97.

[Moll et al., 2010] Moll, J., Schulte, R., Hartmann, B., Fritzen, C.-P., and Nelles, O. (2010). Multi-site damage localization in anisotropic plate-like structures using an active guided wave structural health monitoring system. *Smart Materials and Structures*, 19:045022.

[Myers et al., 2012a] Myers, M. R., Jorge, A. B., Mutton, M. J., and Walker, D. G. (2012a). A comparison of extended Kalman filter, particle filter, and least squares localization methods for a high heat flux concentrated source. *International Journal of Heat and Mass Transfer*, 55(9-10):2219–2228.

[Myers et al., 2012c] Myers, M. R., Jorge, A. B., Mutton, M. J., and Walker, D. G. (2012c). High heat flux point source sensitivity and localization analysis for an ultrasonic sensor array. *International Journal of Heat and Mass Transfer*, 55(9-10):2472–2485.

[Myers et al., 2012d] Myers, M. R., Jorge, A. B., Mutton, M. J., and Walker, D. G. (in prep, 2012d). Hypersonic vehicle boundary layer transition detection using ultrasound and the extended kalman filter. *AIAA Journal*.

[Myers et al., 2012e] Myers, M. R., Jorge, A. B., Mutton, M. J., and Walker, D. G. (in prep, 2012e). Simultaneous localization and parameter identification (SLAP) of a high heat flux concentrated step source. *International Journal of Heat and Mass Transfer*.

[Myers et al., 2012b] Myers, M. R., Jorge, A. B., Mutton, M. J., and Walker, D. G. (in press, 2012b). A comparison of extended Kalman filter ultrasound time-of-flight measurement models for heating source localisation. *Inverse Problems in Science and Engineering*. (in press).

[Myers et al., 2010a] Myers, M. R., Jorge, A. B., Walker, D. G., and Mutton, M. J. (2010a). A comparison of extended Kalman filter approaches using non-linear temperature and ultrasound time-of-flight measurement models for heating source localization of a transient heat transfer problem. In *Inverse Problems, Design and Optimization Symposium*, number IPDO-027, Brazil. IPDO.

[Myers et al., 2010b] Myers, M. R., Jorge, A. B., Walker, D. G., and Mutton, M. J. (2010b). A comparison of extended Kalman filter, extended information filter, and least squares approaches for parameter identification of a transient heat transfer problem. In *Inverse Problems Symposium*, East Lansing, MI. Michigan State University.

[Myers et al., 2011] Myers, M. R., Jorge, A. B., Walker, D. G., and Mutton, M. J. (2011). Heat source localization sensitivity analyses for an ultrasonic sensor array. In *ASME/JSME 2011 8th Thermal Engineering Joint Conference*, number AJTEC2011-44120, Honolulu, HI. ASME.

[Myers et al., 2012g] Myers, M. R., Jorge, A. B., Yuhas, D. E., and Walker, D. G. (2012g). A novel ultrasonic method for locating the boundary layer transition region on a hypersonic vehicle. In *50th AIAA Aerospace Sciences Meeting, January 9-12, 2012, and Nashville, TN*, number AIAA-2012-0859, Nashville, TN. AIAA.

[Myers et al., 2012f] Myers, M. R., Jorge, A. B., Yuhas, D. E., and Walker, D. G. (in press, 2012f). An adaptive extended Kalman filter for localizing a high heat flux point source using an ultrasonic sensor array. *Computer Modeling in Engineering and Sciences*.

[Myers et al., 2008] Myers, M. R., Walker, D. G., Yuhas, D. E., and Mutton, M. J. (2008). Heat flux determination from ultrasonic pulse measurements. In *International Mechanical Engineering Congress and Exposition*, number IMECE2008-69054. ASME.

[Myers et al., 2010c] Myers, M. R., Walker, D. G., Yuhas, D. E., and Mutton, M. J. (in review, 2010c). Heat flux determination from ultrasonic pulse measurements. *International Journal of Heat and Mass Transfer*. (in review).

[Paliwal, K.K. and Basu, Anjan, 1987] Paliwal, K.K. and Basu, Anjan (1987). A speech enhancement method based on Kalman filtering. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '87*, pages 177–180.

[Pullins and Diller, 2010] Pullins, C. and Diller, T. (2010). In situ high temperature heat flux sensor calibration. *International Journal of Heat and Mass Transfer*, 53:3429–3438.

[Recreational Aviation Australia, 2010] Recreational Aviation Australia (2010). Flow regimes. http://www.auf.asn.au. [Online; accessed 10-March-2010].

[Reed et al., 1997] Reed, H. L., Kimmel, R. L., Schneider, S., Arnal, D., and Saric, W. (1997). Drag prediction and transition in hypersonic flow. In *Symposium on Sustained Hypersonic Flight, AGARD Conference on Future Aerospace Technology in the Service of the Alliance*, Palaiseau, France. NATO Research and Technology Organisation.

[Roache, 1998] Roache, P. J. (1998). *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers.

[Rochinha and Peirce, 2010] Rochinha, F. A. and Peirce, A. (2010). Monitoring hydraulic fractures: state estimation using an extended Kalman filter. *Inverse Problems*, 26:1–18.

[Salmond et al., 1993] Salmond, D., Gordon, N., and Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F, Radar and signal processing*, 140(2):107–113.

[Schneider, 1999] Schneider, S. (1999). Flight data for boundary-layer transition at hypersonic and supersonic speeds. *Journal of Spacecraft and Rockets*, 36(1):8–20.

[Schneider, 2004] Schneider, S. (2004). Hypersonic laminar-turbulent transition on circular cones and scramjet forebodies. *Progress in Aerospace Sciences*, 40(1-2):1–50.

[Schook et al., 2001] Schook, R., Lange, H. D., and Steenhoven, A. V. (2001). Heat transfer measurements in transitional boundary layers. *International Journal of Heat and Mass Transfer*, 44(5):1019–1030.

[Tasman et al., 1977] Tasman, H. A., Schmidt, H. E., Richter, J., Campana, M., and Fayl, G. (1977). Treson experiments: Measurement of temperature profiles in nuclear fuels by means of ultrasonic thermometers. *High Temperatures - High Pressures*, 9:387–406.

[Thrun et al., 2006] Thrun, S., Burgard, W., and Fox, D. (2006). *Probabilistic Robotics*. The MIT Press, Cambridge, MA.

[Turcotte and Schubert, 2002] Turcotte, D. L. and Schubert, G. (2002). *Geodynamics*. Cambridge University Press.

[Vianna et al., 2009] Vianna, F., Orlande, H., and Dulikravich, G. (2009). Prediction of the temperature field in pipelines with Bayesian filters and non-intrusive measurements. In *Proceedings of the 20th International Congress of Mechanical Engineering*, Gramado, RS, Brazil.

[Vianna et al., 2010] Vianna, F. L. V., Orlande, H., and Dulikravich, G. (2010). Temperature field prediction of a multilayered composite pipeline based on the particle filter method. *Proceedings of the 14th International Heat Transfer Conference*.

[Wadley et al., 1986] Wadley, H. N. G., Norton, S. J., Mauer, F., Droney, B., Ash, E. A., and Sayers, C. M. (1986). Ultrasonic measurement of internal temperature distribution. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 320(1554):341–361.

[Walker et al., 2000] Walker, D., Scott, E., and Nowak, R. (2000). Estimation methods for two-dimensional conduction effects of shock-shock heat fluxes. *Journal of Thermophysics and Heat Transfer*, 14(4):533–539.

[Woodbury, 2003a] Woodbury, K. A., editor (2003a). *Inverse Engineering Handbook*. CRC Press, Boca Raton, FL.

[Woodbury, 2003b] Woodbury, K. A. (2003b). Sequential function specification method using future times for function estimation. In Woodbury, K. A., editor, *Inverse Engineering Handbook*. CRC Press, Boca Raton, FL.

[Yee and Couchman, 1976] Yee, B. G. W. and Couchman, J. C. (1976). Application of ultrasound to NDE of materials. *IEEE Transactions on Sonics and Ultrasonics*, SU-23(5):299–305.

[Yu et al., 2010] Yu, M., Sarner, G., C.C.M.Luijten, Alden, M., Baert, R., and de Goey, L. (2010). Survivability of thermographic phosphors (YAG:Dy) in a combustion environment. *Measurement Science and Technology*, 21:037002.

[Zhu et al., 2010] Zhu, X. P., Rizzo, P., Marzani, A., and Bruck, J. (2010). Ultrasonic guided waves for nondestructive evaluation/structural health monitoring of trusses. *Measurement Science and Technology*, 21(4):045701.